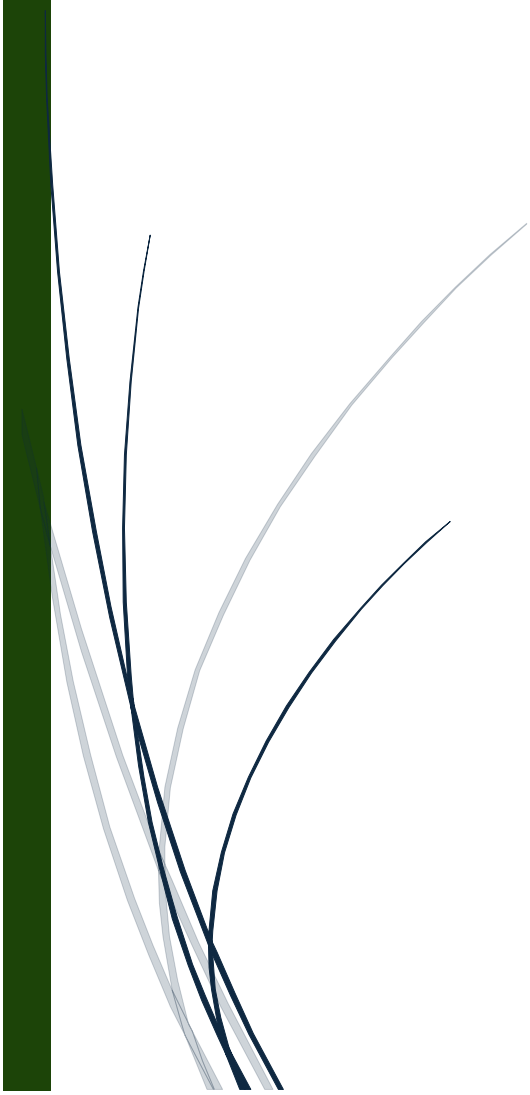
12/12/2024

Forager Design Document



CSE 430-01 December 2024

Contents

Document Information

Scope

This is a system which facilitates a user in finding edible plants near their location. The system consists of a server containing a fixed set of plans, and a mobile client which allows the user to view the list of plants. The server consists of a database containing the collection of plants, and an administrator console which allows an administrator to modify the database, and an expert to add/change entries in the database.

Authorship

Project Manager
- Daniel Malasky

Team Kelly: Phone Services
- Kelly Payne, Carson Jones, Bryce Chesley

Team Emilio: UI
- Emilio Ordonez Guerrero, Ashley DeMott, Wade Withers, Joseph Linton

Team Joseph: Server
- Joseph Gyman, Colby Hale, Gabe Hayes, Brock Hoskins

References

Software Requirements Specification: Outlines requirements for the Forager App. SRS-Forage

Change History

The latest version of this document was released on 11/21/2024. The final version is set to be released on 12/17/2024.

Version 0.1
- Added View 1, the Overall Application View

Version 0.2
- Adjusted View 1, added views 1.1a, 1.1.2a, 1.1.2b, 1.2.3, 1.2.1a, 1.2.1b, 1.2.2, and 1.2.1.1

Version 0.3
- Adjusted View 1.2.3, added Front Matter, updated “Reference By” sections of all views, added views 1.2.1.1.2, 1.1.2.1, 1.1.2.4a, 1.1.2.4b, 1.1.1.1, 1.1.1.2
- Adjusted 1.2.1.1 to include JWT login session token

Version 0.4
- Added views 1.1.2.5, 1.1.1.6, 1.1.1.6b, 1.1.2.2, changed references of mySQL to postgreSQL, adjusted views 1.1a, 1.1.2b, 1.1.1.1, 1.1.1.2, 1.1.1.3, 1.1.2.0, 1.1.2.4b, 1.2.3.4a, 1.2.3.4b, 1.2.3.2, 1.2.3.1, 1.2.1c, 1.2.1d, 1.2.1e, 1.2.1.1a-1.2.1.1d and 1.1.2.5

- Removed Email verification from 1.2.1.3 view

Version 0.5

- Adjusted views 1.2.1.1a, 1.2.1.7, 1.2.3.4, 1.2.1.13, 1.1.1.6a, 1.1.1.6b, 1.1.1.5, 1.1.1.1, 1.1.1.2, 1.1.1.3, 1.1.2.4b, 1.2.1.7.3, 1.2.1a

Stakeholders

Sponsor: James Helfrich

Project Manager: Daniel Malasky

Chief Technology Expert: Brock Hoskins

Chief of Quality Assurance: Ashley DeMott

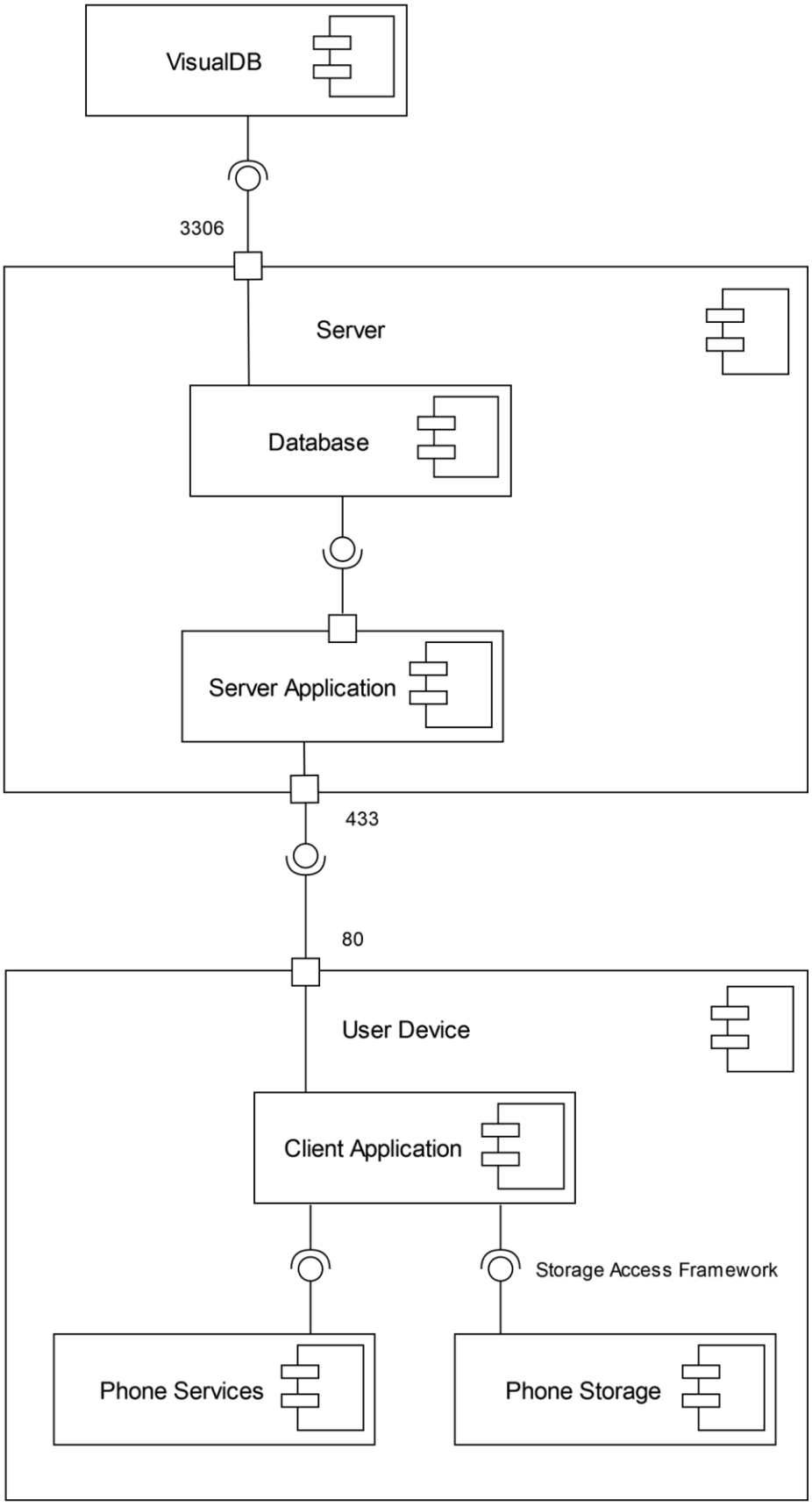
Chief Editor: Carson Jones

Viewpoints

Component Diagram	Helfrich, . Software Design. Available from: Brigham Young University - Idaho, (2nd Edition). Kendall Hunt Publishing, 2024.
Dataflow Diagram	Helfrich, . Software Design. Available from: Brigham Young University - Idaho, (2nd Edition). Kendall Hunt Publishing, 2024.
Entity Relationship Diagram	“MySQL :: MySQL Workbench Manual :: 9.1.1.3 EER Diagrams.” Dev.mysql.com, dev.mysql.com/doc/workbench/en/wb-eer-diagrams-section.html.
String	“Kotlin Strings.” Www.w3schools.com, www.w3schools.com/KOTLIN/kotlin_strings.php.
Pseudocode	Helfrich, . Software Design. Available from: Brigham Young University - Idaho, (2nd Edition). Kendall Hunt Publishing, 2024.
Class Diagram	Helfrich, . Software Design. Available from: Brigham Young University - Idaho, (2nd Edition). Kendall Hunt Publishing, 2024.
JSON Schema	“JSON Schema - What Is a Schema?” Json-Schema.org, 2020, json-schema.org/understanding-json-schema/about. Accessed 22 Nov. 2024.
Site Map	How. “How to Create a Sitemap Using Draw.io (Diagrams.net).” YouTube, 20 Oct. 2020, youtu.be/VGEqKIt03aw?si=wos11Oqa6580-3FM. Accessed 22 Nov. 2024.
Flowchart	Helfrich, . Software Design. Available from: Brigham Young University - Idaho, (2nd Edition). Kendall Hunt Publishing, 2024.

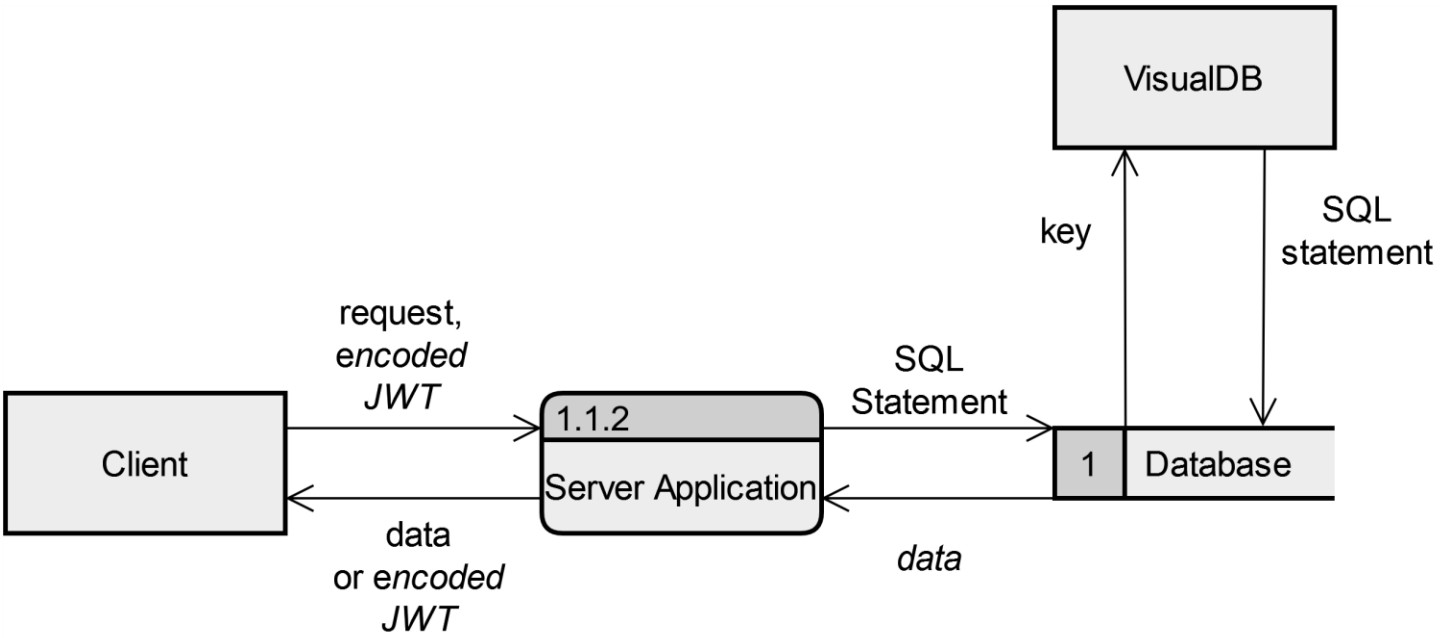
Views

View 1: Overall Application View



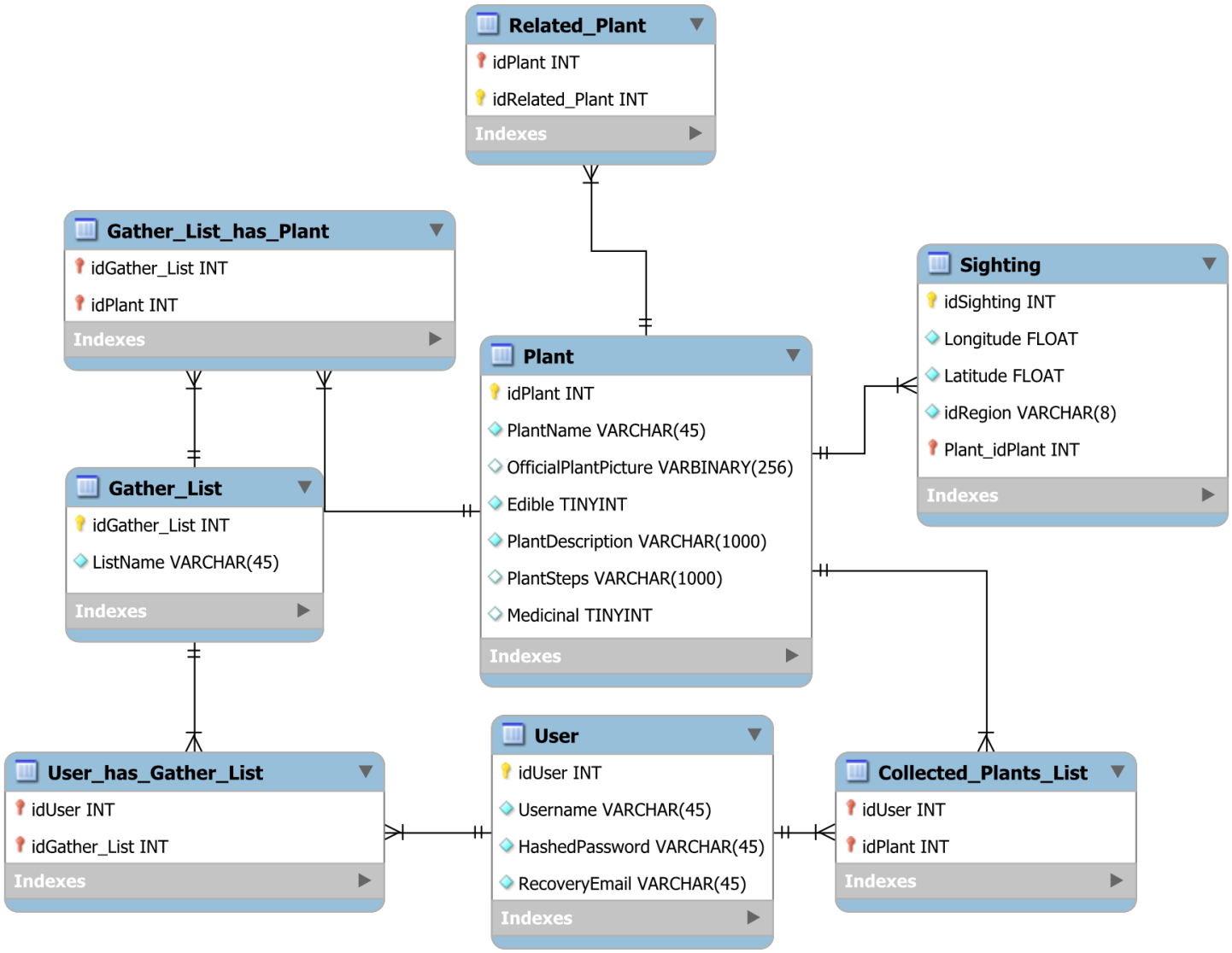
Name	#1 Overall Application View
Purpose	An overall view of the entire application and its relationship to the **Server** and **Client**.
Description	Shows the different components necessary for the application and their interactions with other components or parent components.
Requirements	The current diagram fulfills every requirement in the SRS.
Elements	**1.1 Server:** The back-end part of the **Client-Server** pair. The **Server** connects with the **App.**
	1.1.1 Database: A collection of **Plant** information and **User Credentials**.
	1.1.2 Server Application: Responsible for managing the core logic of the **Server.**
	VisualDB: An external service providing a user interface for an **Administrator** to modify the **Database.** (https://visualdb.com/docs/index.html)
	1.2 User Device: The device that runs the **App** and stores user information.
	1.2.1 Client Application: Serves as the front-end interface through which the **User** interacts.
	1.2.2 Phone Services: Device-specific components, which are the GPS and camera. The camera will be accessed through the Camera2 API under android.hardware.camera2 (https://developer.android.com/media/camera/camera2). The GPS will be accessed through the Google Play Services Location API (https://developers.google.com/android/reference/com/google/android/gms/location/LocationServices).
	1.2.3 Phone Storage: Storage on the user's device through media store (https://developer.android.com/training/data-storage/shared/media).(https://developer.android.com/training/data-storage)
Referenced By	
Viewpoint	Component Diagram

View 1.1: Server View



Name	#1.1 Server View
Purpose	To show how data flows throughout the server at the highest level view.
Description	Requests are sent from the client to the server application which interacts with the **Database**. It will send back either a **Plant** list or JWT based on the request. VisualDB is used as an **Administrator Console** to edit the database.
Requirements	8, 9, 22, 23
Elements	**1.2.1 Client**: The application on a mobile platform (Android).
	1.1.2 Server Application: The part of the **Server** that responds to and processes requests from the **Client**.
	1.1.1 Database: The list of user **Credentials** as well as **Plant** data.
	Encoded JWT: An encoded string used to authenticate the **User**. This is the form of the JWT in transit, and when sent to the **Client**.
	1.1.1.6 SQL Query: A request used to retrieve or modify the **Database**.
	1.1.1.6b Data: A JSON of the data returned from the **Database**.
	1.1.3 VisualDB: A 3rd party tool to edit the **Database**. Outside of the scope of this design. (https://visualdb.com/docs/index.html)
Referenced By	**1 Overall Application View, 1.2.1.7.2 Get Local Plants View**
Viewpoint	Data Flow Diagram

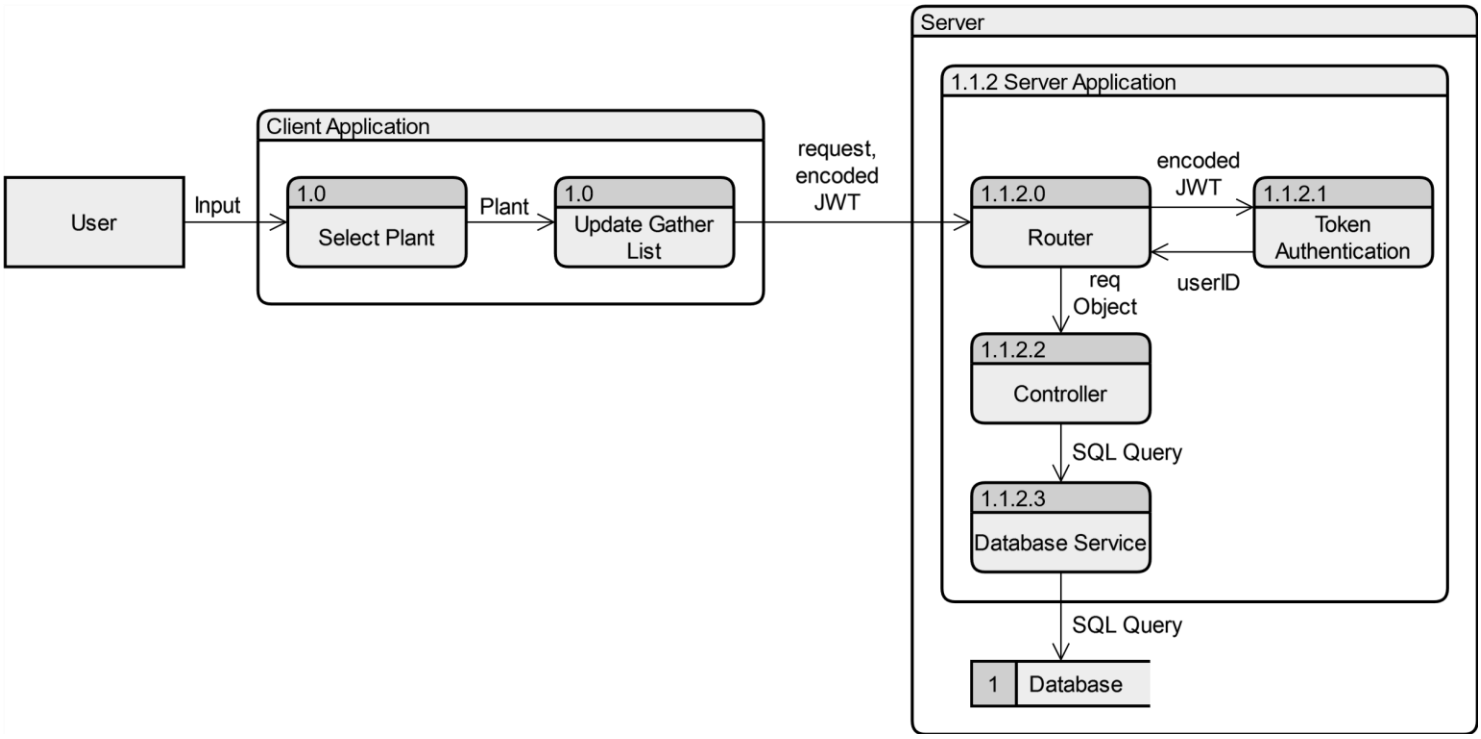
View 1.1.1: Database View



Name	#1.1.1 Database View
Purpose	Show the structure of the database
Description	An entity relationship diagram to show the storage methods and relationships between all data stored within the **Database**.
Requirements	7-12
Elements	**Plant:** Stores most of the **Plant's** information that the **User** will need, including the **Name**, **Picture**, **Use, Description,** and **Steps**.
	idPlant: A unique, auto-generated, and auto-incremented integer used to identify an individual **Plant**.
	PlantName: The name of a **Plant**.
	OfficialPlantPicture: A **Plant** picture to visually represent a **Plant**. Stored as a byte array.
	Edible: A boolean identifying whether the **Plant** is safely edible. Represented as 0 or 1.
	PlantDescription: A brief description of the **Plant**.
	PlantSteps: A brief explanation of the use of the **Plant**.
	Medicinal: A boolean identifying whether the **Plant** has medicinal use. Represented as 0 or 1.
	Related_Plant: Stores the combination of id's of every **Plant** and their related **Plants**.
	idRelated_Plant: The id of a related **Plant**.
	User: Stores the **User's** unique id, username, password, and email.
	idUser: A unique, auto-generated, and auto-incremented integer used to identify an individual **User**.
	Username: A **User's** username.
	HashedPassword: The hashed version of a **User's** password.
	RecoveryEmail: A **User's** recovery email.

	Gather_List_has_Plant: A join table that stores the collection of key combinations between idGather_list and idPlant
	Gather_List: Stores the **Gather List's** unique id and its **Name**.
	idGather_List: A unique, auto-generated, and auto-incremented integer used to identify an individual **Gather List**.
	ListName: The name of a **Gather List**.
	User_has_Gather_List: A join table that stores the key combinations between idGather_List and idUser
	Collected_Plants_List: A join table that stores the collection of key combinations between idUser and idPlant.
	Sighting: Stores the geographic location of a specific sighting, the associated **Region** of a sighting, and the plant that was sighted.
	idSighting: A unique, auto-generated, and auto-incremented integer used to identify an individual **User**.
	Longitude: A float representing the specific longitude of a sighting.
	Latitude: A float representing the specific latitude of a sighting.
	idRegion: A combination of the rounded latitude and longitude of the sighting to organize it into a region. Stored as a string. Example: *-30, 25*
Referenced By	**1 Overall Application View, 1.1 Server View, 1.1.2b Server Application View, 1.1.1.1 Update Gather List View, 1.1.1.2 Checking off From Gather List View, 1.1.2.5 User Sign Up View, 1.1.2.1 Token Authentication Class View**
Viewpoint	Entity Relationship Diagram

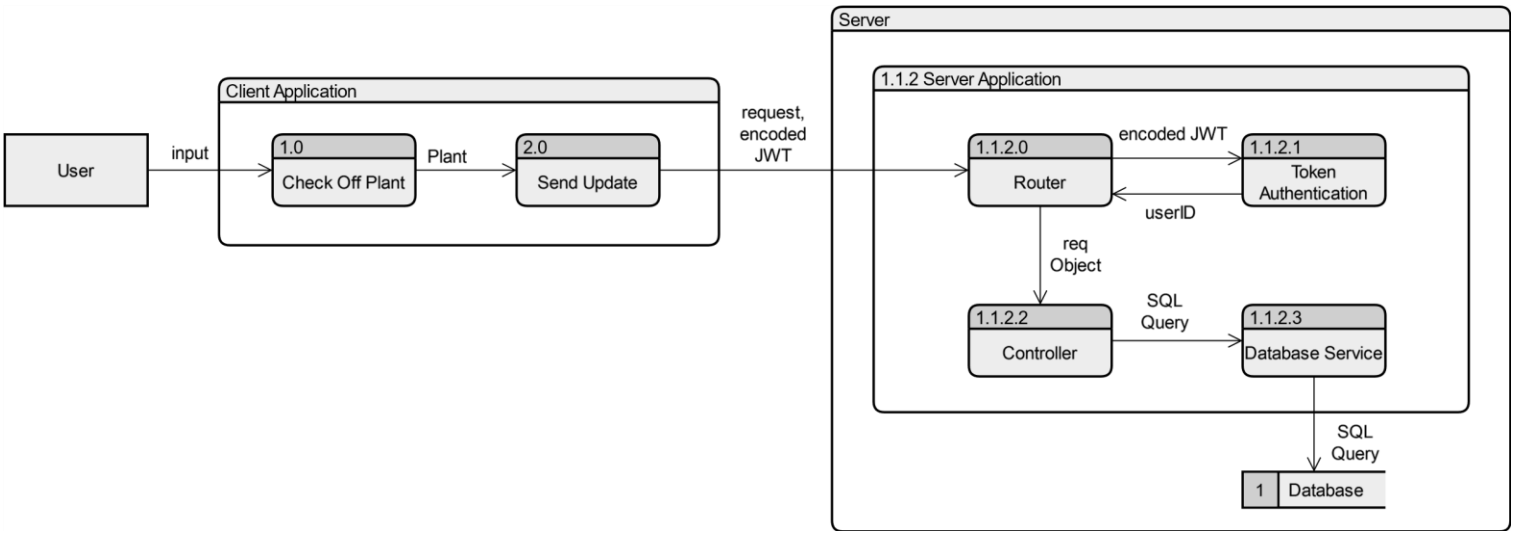
View 1.1.1.1 Update Gather List View



Name	#1.1.1.1 Update Gather List View
Purpose	The purpose of this data flow diagram is to illustrate the sequence of data interactions when a **User** adds to or removes from the **Gather List**.
Description	Starting with the **User**'s selection of a specific **Plant**, the **User**'s selection flows through the **App** and then it is sent to the **Server** to be added/removed from the **Gather List**.
Requirements	17, 18
Elements	**Select Plant**: The **User** interacts with the **App** to select a **Plant** from the **All Plants/Local Plants List.**
	Update Gather List: The **App** sends a request to the **Server** to either add or remove the selected **Plant** to/from the **Gather List**.
	1.1.2 ServerApplication: the part of the **Server** that responds to and processes requests from the **client**.
	1.1.1 Database: the list of **user credentials** as well as **plant** data.
	1.1.2.0 Router: The receiver of requests and JWT from the **client**.
	1.1.2.3 Encoded JWT: An encoded string used for authentication.
	1.1.1 userID: The **user**'s Identification extracted from the Unencoded JWT Object in the form of an integer.
	1.1.1.6a SQL Query: A request used to retrieve or modify the **database**.
	Request: An http message sent from the **Client.** (https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages)
	Req Object: A object that is made from the request that includes the userID as an attribute (https://expressjs.com/en/api.html#req)
	1.1.2.2 Controller: Returns responses to the **client** and decides how requests should be handled.
	1.1.2.3 Database Service: Retrieves **plant** data from the **database**.
	1.1.2.1 Token Authentication: Confirms that the JWT is authentic.
	1.1 Server: The back-end part of the **client-server** pair. The **Server** connects with the **App**.

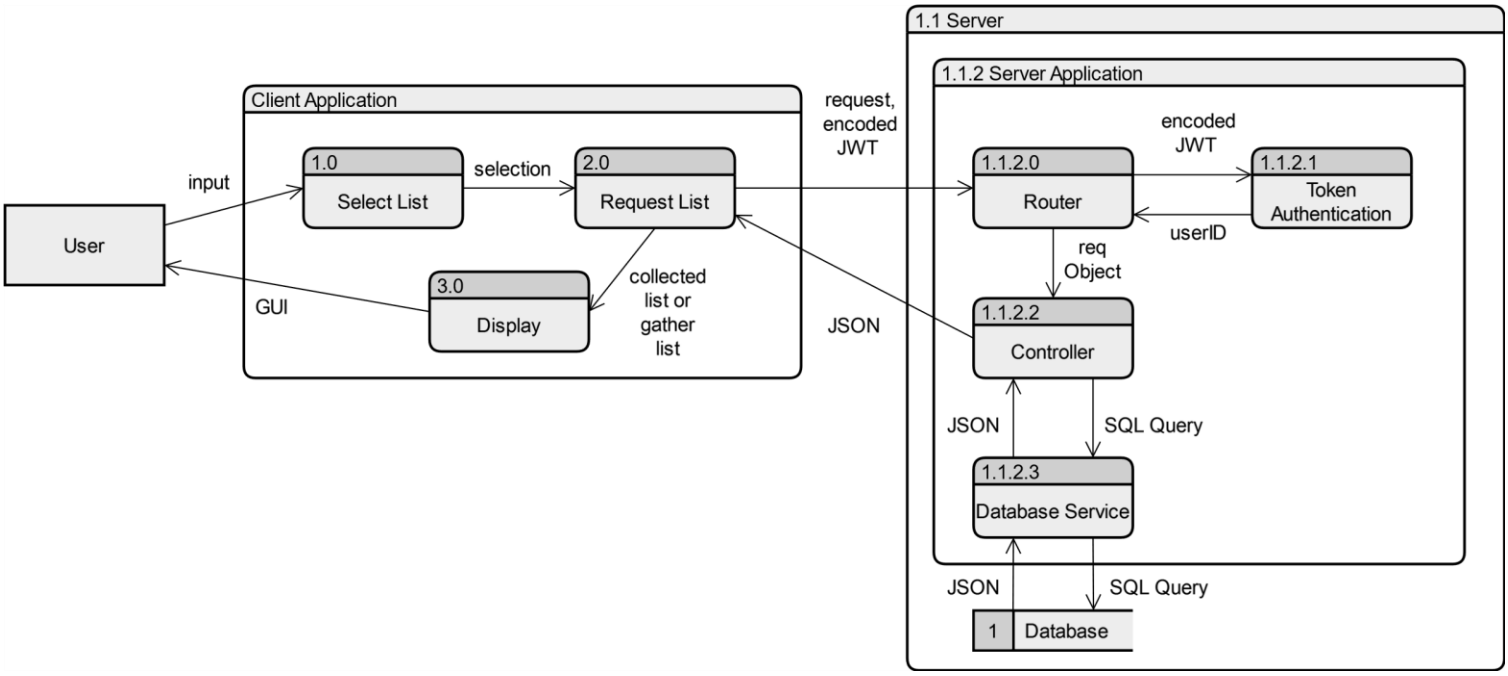
Referenced By	**1.2.1b Home Flowchart View**
Viewpoint	Data Flow Diagram

View 1.1.1.2 Checking Off From Gather List View



Name	#1.1.1.2 Check Off Plant in Gather List View
Purpose	To illustrate the sequence of data interactions when a **User** checks off a **Plant** in the **Gather List** within the **App**.
Description	Starting with the **User's** selection of a specific **Plant** in the **Gather List**, the **User's** selection flows through the **App** and then it is sent to the **Server** to be marked off and subsequently added to the **Collected Plants List**.
Requirements	15, 19
Elements	**Check Off Plant:** The **User** interacts with the **App** to select a **Plant** from the **Gather List**.
	Send Update: The **App** sends a request to the **Server** to update the **Gather List** in local storage by adding the **Plant**.
	1.1.2 ServerApplication: The part of the **Server** that responds to and processes requests from the **Client**.
	1.1.2.0 Router: The receiver of requests and JWT from the **Client**.
	1.1.2.3 Encoded JWT: An encoded string used to authenticate the **User.** This is the form of the JWT in transit, and when sent to the **Client**.
	1.1.1 userID: The **user**'s Identification extracted from the Unencoded JWT Object in the form of an integer.
	Request: An http message sent from the Client. (https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages)
	Req Object: A object that is made from the request that includes the userID as an attribute (https://expressjs.com/en/api.html#req)
	1.1.1.6a SQL Query: A request used to retrieve or modify the **Database**.
	1.1.2.2 Controller: Returns responses to the **Client** and decides how requests should be handled.
	1.1.2.3 Database Service: Retrieves **Plant** data from the **Database**.
	1.1.2.1 Token Authentication: Confirms that the JWT is authentic.
Referenced By	**1.2.1b Home Flowchart View**
Viewpoint	Data Flow Diagram

View 1.1.1.3 Get Gather/Collected List View



Name	#1.1.1.3 Get Gather/Collected List View
Purpose	To illustrate the sequence of data interactions when a **User** wants to view their **Gather List** or their **Collected Plant List**.
Description	Starting with the **User's** selection to view either their **Gather List** or their **Collected Plant List**. The **App** then sends a request to the **Server** to retrieve the selected list.
Requirements	14, 16
Elements	**Select List:** The **User** interacts with the **App** to select either their **Gather List** or their **Collected Plant List**.
	Request List: The **App** sends a request to the **Server** to retrieve the selected list.
	1.1.2 Server Application: The part of the **Server** that responds to and processes requests from the **Client**.
	1.1.2.0 Router: The receiver of requests and JWT from the **client**.
	1.1.1 userID: The **user**'s Identification extracted from the Unencoded JWT Object in the form of an integer.
	JSON: A json file.
	Request: An http message sent from the Client. (https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages)
	Req Object: A object that is made from the request that includes the userID as an attribute (https://expressjs.com/en/api.html#req)
	1.1.2.3 Encoded JWT: An encoded string used to authenticate the **User.** This is the form of the JWT in transit, and when sent to the **Client**.
	1.1.2.3 Unencoded JWT Object: The full contents of the JWT in its unencoded form
	1.1.1.6a SQL Query: A request used to retrieve data or modify the **Database**.
	1.1.2.2 Controller: Returns responses to the **Client** and decides how requests should be handled.
	1.1.2.3 Database Service: Retrieves **Plant** data from the **Database**.
	1.1.2.1 Token Authentication: Confirms that the JWT is authentic.
	Display: The **App** displays the selected list to the **User**.

	GUI: The graphical user interface displayed to the **User.**
Referenced By	**1.2.1b Home Flowchart View**
Viewpoint	Data Flow Diagram

View 1.1.1.5 Hashed Password View

Plaintext Password:

“MyP@ssword1”

Bcrypt Hashed Password:

$2y$10$ncIPB5Poega0ly1GF8J72e7QXGQ7WpUIZTqlMGG6KGP.mVYKumOVO

Pseudocode:

```
bcrypt.hashSync(plaintextPassword, 10)
```

Name	#1.1.1.5 Hashed Password View
Purpose	Defines a hashed password with an example using Bcrypt algorithm.
Description	A plaintext password will be turned into a hashed password using the common Bcrypt algorithm for security. This hashed password will then be stored in the **Database** as a **User's** password credential. (https://www.npmjs.com/package/bcrypt)
Requirements	23
Elements	**Plaintext Password:** The example password in plaintext before the hashing.
	Bcrypt: A hashing algorithm used to hash passwords.
	hashSync(): A function that hashes a password with a given amount of rounds of salt.
	Hashed Password: The example password post-hash to be stored in the **Database**.
Referenced By	**1.1.1 Database View, 1.1 Server View, 1.1.2b Server Application View,**
Viewpoint	Pseudocode/String

View 1.1.1.6a Database Queries View

1.1.1.6.1a getUserByID

getUserById(id)

```
data <- database.query("SELECT * FROM User WHERE idUser = {id}")
RETURN data.rows[0]
```

1.1.1.6.2a updatePassword

updatePassword(id,newPassword)

```
data <- database.query("UPDATE User SET HashedPassword = {newPassword} WHERE idUser = {id}")
RETURN data.rows[0]
```

1.1.1.6.3a updateUsername

updateUsername(id, newUsername)

```
data <- database.query("UPDATE User SET Username = {newUsername} WHERE idUser = {id}")
RETURN data.rows[0]
```

1.1.1.6.4a addGatherList

addGatherlist(userId, gatherListName)

```
gatherlist_data <- database.query("INSERT INTO Gather_List (ListName) VALUES ({gatherListName})")
RETURN database.query("INSERT INTO User_has_Gather_List (idUser, idGather_List) VALUES ({userId}, {gatherlist_data.rows[0]["idGather_List"]})")
```

1.1.1.6.5a addPlantToGatherList

addPlantToGatherList(gatherListId, plantId)

```
RETURN database.query("INSERT INTO Gather_List_has_Plant (idGather_List, idPlant) VALUES ({gatherListId}, {plantId})")
```

1.1.1.6.6a removePlantFromGatherList

removePlantFromGatherList(gatherListId, plantId)

```
data <- database.query("DELETE FROM Gather_List_has_Plant WHERE idGather_List = {gatherListId} AND idPlant = {plantId}")
```

```
    RETURN data
```

1.1.1.6.7a addPlantToCollectedList

addPlantToCollectedList(userId, plantId)

```
    RETURN database.query("INSERT INTO Collected_Plants_List (idUser, idPlant)
VALUES({userId}, {plantId})")
```

1.1.1.6.8a getGatherListByUserId

getGatherListByUserId(userId)

```
    data <- database.query("SELECT ListName, idGather_List, idPlant, PlantName
FROM Gather_List
    INNER JOIN Gather_List_has_Plant on Gather_List.idGather_List =
Gather_List_has_Plant.idGather_list
    INNER JOIN Plant on Gather_List_has_Plant.idPlant = Plant.idPlant
    INNER JOIN User_has_Gather_List on Gather_List.idGather_List =
User_has_Gather_List.idGather_List
    WHERE User_has_Gather_List.idUser = {userId}")
    RETURN data.rows
```

1.1.1.6.9a getGatherListById

getGatherListById(id)

```
    data <- database.query("SELECT ListName, PlantName, idPlant FROM
Gather_List_has_plant
    INNER JOIN Plant on Gather_List_has_Plant.idPlant = Plant.idPlant
    WHERE idGather_List = {id}")
    RETURN data.rows
```

1.1.1.6.10a getCollectedList

getCollectedList(userId)

```
    data <- database.query("SELECT PlantName, idPlant FROM Collected_Plants_List
FROM
    INNER JOIN Plant on Collected_Plants_List.idPlant = Plant.idPlant
    WHERE idUser = {userId}")
    RETURN data.rows
```

1.1.1.6.11a removePlantFromCollectedList

removePlantFromCollectedList(userId, plantId)

```
    data <- database.query(“DELETE FROM Collected_Plants_List WHERE idUser= {userId} AND idPlant = {plantId}”)
    RETURN data
```

1.1.1.6.12a getPlantById

getPlantById(id)

```
    data <- database.query(“SELECT * FROM Plant WHERE idPlant = {id}”)
    RETURN data.rows[0]
```

1.1.1.6.13a getUniquePlantIDsByRegion

getUniquePlantIDsByRegion(regionID)

```
    data <- database.query("SELECT DISTINCT Plant_idPlant FROM Sighting WHERE Region_idRegion = {regionID}")
    return data.rows
```

1.1.1.6.14a getAllPlants

getAllPlants()

```
    data <- database.query(“SELECT idPlant, plantName FROM Plant”)
    RETURN data.rows
```

1.1.1.6.15a getAccountByUsername

getAccountByUsername(username)

```
     data <- database.query(“SELECT * FROM User WHERE Username = {username}”)
     RETURN data.rows
```

1.1.1.6.16a getSightingsByPlantAndRegion

getSightingsByPlantAndRegion(regionID, plantID)

```
    data <- database.query("SELECT Longitude, Latitude FROM Sighting WHERE
Plant_idPlant = {plantID} AND Region_idRegion = {regionID}")
    return data.rows
```

Name	#1.1.1.6a Database Queries View
Purpose	Show what specific queries will need to be made to retrieve the right data from the **Database**.
Description	There are several queries for adding, removing, and retrieving information from **Gather Lists** as well as registering **Users** and updating their information.
Requirements	3, 6, 7, 10, 11, 13-19
Elements	**1.1.1 Database**: A view of the **Database** as a whole, including the data within and how it is stored.
	1.1.1.6.1a getUserById: Retrieves all the information from a **User** by the specified id.
	1.1.1.6.2a updatePassword: Updates the **User's** password.
	1.1.1.6.3a updateUsername: Updates the **User's** username.
	1.1.1.6.4a addGatherList: Inserts the new **Gather List's** name into the **Gather List** table, saves the last id from that table, then attaches that id with the specified user id in the User_has_Gather_List table.
	1.1.1.6.5a addPlantToGatherList: Inserts the id of a given **Plant** to a given **Gather List.**
	1.1.1.6.6a removePlantFromGatherList: Removes a given **Plant** from a given **Gather List.**
	1.1.1.6.7a addPlantToCollectedList: Adds a given **Plant's** id to the user's **Collected List.**
	1.1.1.6.8a getGatherListByUserId: Retrieves the name of the **Gather Lists** and their respective **Plant** information based on the given user id.
	1.1.1.6.9a getGatherListById: Retrieves the name of a **Gather List** and its respective **Plant** information based on the given **Gather List** id.
	1.1.1.6.10a getCollectedList: Retrieves the list of **Plants** that belong to the **Collected List** based on the given **User** id.
	1.1.1.6.11a removePlantFromCollectedList: Removes a given **Plant's** id based on the given **User** id.
	1.1.1.6.12a getPlantById: Retrieves the information from one **Plant** based on the given **Plant** id.
	1.1.1.6.13a getUniquePlantIDsByRegion: Retrieves all unique **Plant** ids from Sighting table by **Region**.

	1.1.1.6.14a getAllPlants: Retrieves the name and id of every **Plant**.
	1.1.1.6.15a getAccountByUsername: Retrieves the **User's Credentials** by the **Username**.
	1.1.1.6.16a getSightingsByPlantAndRegion: Retrieves all **Locations** from Sighting table, for a specific **Plant** in a specific **Region**.
Referenced By	**1 Overall Application View, 1.1 Server View, 1.1.1.1 Update Gather List View, 1.1.1.2 Checking Off From Gather List View, 1.1.1.3 Get Gather/Collected List View, 1.1.2.0b Router Pseudocode View, 1.1.2.2 Controller Pseudocode View**
Viewpoint	Pseudocode

View 1.1.1.6b Database Data JSON View

1.1.1.6.1b getUserById/1.1.1.6.2b getUserByUsername

getUserById/getUserByUsername


```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
    "rowCount": {
      "type": "integer"
    },
    "oid": {
      "type": "integer"
    },
    "rows": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "Username": {
              "type": "string"
            },
            "HashedPassword": {
              "type": "string"
            },
            "RecoveryEmail": {
              "type": "string"
            },
            "idUser": {
              "type": "string"
            }
          },
          "required": [
            "Username",
            "HashedPassword",
            "RecoveryEmail",
            "idUser"
          ]
        }
      ]
    },
    "fields": {
      "type": "array",
      "items": [
        {
          "type": "object",
```

```
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
```

```
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "SELECT",
     "rowCount": 1,
     "oid": 0,
     "rows": [ {
                "Username": "example",
                "HashedPassword": "example",
                “RecoveryEmail”: "example",
                “idUser”: "example"
              } ],
     "fields": [
                {
                "name": “Username”,
                "dataTypeID": 1043
                },
                {
                "name": "HashedPassword",
                "dataTypeID": 1043
                },
                {
                "name": "RecoveryEmail",
                "dataTypeID": 1043
                },
                "name": "idUser",
                "dataTypeID": 23
                }]
}
```

1.1.1.6.3b updatePassword

updatePassword

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
    "rowCount": {
      "type": "integer"
    },
    "oid": {
      "type": "integer"
    },
    "rows": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "HashedPassword": {
              "type": "string"
            }
          },
          "required": [
            "HashedPassword"
          ]
        }
      ]
    },
    "fields": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
```

```
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "UPDATE",
     "rowCount": 1,
     "oid": 0,
     "rows": [ {
                "HashedPassword": "example",
              } ],
     "fields": [
                {
                "name": “HashedPassword”,
                "dataTypeID": 1043
                }]
}
```

1.1.1.6.4b updateUsername

updateUsername

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
    "rowCount": {
      "type": "integer"
    },
    "oid": {
      "type": "integer"
    },
    "rows": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "Username": {
              "type": "string"
            }
          },
          "required": [
```

```
            "Username"
          ]
        }
      ]
    },
    "fields": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "UPDATE",
     "rowCount": 1,
     "oid": 0,
     "rows": [ {
                "Username": "example",
              } ],
     "fields": [
                {
                "name": “Username”,
                "dataTypeID": 1043
                }]
}
```

1.1.1.6.5b addGatherList

addGatherlist

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
    "rowCount": {
      "type": "integer"
    },
    "oid": {
      "type": "integer"
    },
    "rows": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "idUser": {
              "type": "string"
            },
            "idGather_List": {
              "type": "string"
            }
          },
          "required": [
            "idUser",
            "idGather_List"
          ]
        }
      ]
    },
    "fields": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
```

```
            "dataTypeID"
          ]
        },
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "INSERT",
     "rowCount": 1,
     "oid": 0,
     "rows": [ {
               "idUser": "example",
               "idGather_List": "example"
             } ],
     "fields": [
               {
               "name": “idUser”,
               "dataTypeID": 23
               },
               {
               "name": "idGather_List",
               "dataTypeID": 23
               }]
}
```

1.1.1.6.6b addPlantToGatherList

addPlantToGatherList

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
    "rowCount": {
      "type": "integer"
    },
    "oid": {
      "type": "integer"
    },
    "rows": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "idGather_List": {
              "type": "string"
            },
            "idPlant": {
              "type": "string"
            }
          },
          "required": [
            "idGather_List",
            "idPlant"
          ]
        }
      ]
    },
    "fields": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
```

```
            "dataTypeID"
          ]
        },
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "INSERT",
     "rowCount": 1,
     "oid": 0,
     "rows": [ {
               "idGather_List": "example",
               "idPlant": "example",
             } ],
     "fields": [
               {
               "name": “idGather_List”,
               "dataTypeID": 23
               },
               {
               "name": "idPlant",
               "dataTypeID": 23
               }]
}
```

1.1.1.6.7b removePlantFromGatherList

removePlantFromGatherList

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
    "rowCount": {
      "type": "integer"
    },
    "oid": {
      "type": "integer"
    },
    "rows": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "idGather_List": {
              "type": "string"
            },
            "plantId": {
              "type": "string"
            }
          },
          "required": [
            "idGather_List",
            "plantId"
          ]
        }
      ]
    },
    "fields": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
```

```
            "dataTypeID"
          ]
        },
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "DELETE",
     "rowCount": 1,
     "oid": 0,
     "rows": [ {
               "idGather_List": "example",
               "plantId": "example"
             } ],
     "fields": [
               {
               "name": “idGather_List”,
               "dataTypeID": 23
               },
               {
               "name": "plantId",
               "dataTypeID": 23
               }]
}
```

1.1.1.6.8b addPlantToCollectedList

addPlantToCollectedList

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
    "rowCount": {
      "type": "integer"
    },
    "oid": {
      "type": "integer"
    },
    "rows": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "idUser": {
              "type": "string"
            },
            "idPlant": {
              "type": "string"
            }
          },
          "required": [
            "idUser",
            "idPlant"
          ]
        }
      ]
    },
    "fields": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
```

```
            "dataTypeID"
          ]
        },
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "INSERT",
     "rowCount": 1,
     "oid": 0,
     "rows": [ {
               "idUser": "example",
               "idPlant": "example"
             } ],
     "fields": [
               {
               "name": “idUser”,
               "dataTypeID": 23
               },
               {
               "name": "idPlant",
               "dataTypeID": 23
               }]
}
```

1.1.1.6.9b getGatherListByUserId

getGatherListByUserId

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
    "rowCount": {
      "type": "integer"
    },
    "oid": {
      "type": "integer"
    },
    "rows": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "ListName": {
              "type": "string"
            },
            "id_GatherList": {
              "type": "string"
            },
            "idPlant": {
              "type": "string"
            },
            "plantName": {
              "type": "string"
            }
          },
          "required": [
            "ListName",
            "id_GatherList",
            "idPlant",
            "plantName"
          ]
        }
      ]
    },
    "fields": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "name": {
```

```
      "type": "string"
    },
    "dataTypeID": {
      "type": "integer"
    }
  },
  "required": [
    "name",
    "dataTypeID"
  ]
},
{
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "dataTypeID": {
      "type": "integer"
    }
  },
  "required": [
    "name",
    "dataTypeID"
  ]
},
{
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "dataTypeID": {
      "type": "integer"
    }
  },
  "required": [
    "name",
    "dataTypeID"
  ]
},
{
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "dataTypeID": {
      "type": "integer"
    }
```

```
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "SELECT",
     "rowCount": #,
     "oid": 0,
     "rows": [ {
               "ListName": "example",
               "id_GatherList": "example",
               “idPlant”: "example",
               “plantName”: "example"},
               {...}
             ],
     "fields": [
               {
               "name": “ListName”,
               "dataTypeID": 1043
               },
               {
               "name": "id_GatherList",
               "dataTypeID": 23
               },
               {
               "name": "idPlant",
               "dataTypeID": 23
               },
               {
               “Name”: “plantName”,
               “dataTypeID”: 1043}]}
```

1.1.1.6.10b getGatherListById

getGatherListById

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
```

```
"type": "object",
"properties": {
  "command": {
    "type": "string"
  },
  "rowCount": {
    "type": "integer"
  },
  "oid": {
    "type": "integer"
  },
  "rows": {
    "type": "array",
    "items": [
      {
        "type": "object",
        "properties": {
          "ListName": {
            "type": "string"
          },
          "id_GatherList": {
            "type": "string"
          },
          "idPlant": {
            "type": "string"
          },
          "plantName": {
            "type": "string"
          }
        },
        "required": [
          "ListName",
          "id_GatherList",
          "idPlant",
          "plantName"
        ]
      }
    ]
  },
  "fields": {
    "type": "array",
    "items": [
      {
        "type": "object",
        "properties": {
          "name": {
            "type": "string"
          },
          "dataTypeID": {
            "type": "integer"
```

```
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
```

```
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "SELECT",
     "rowCount": #,
     "oid": 0,
     "rows": [ {
               "ListName": "example",
               "id_GatherList": "example",
               “idPlant”: "example",
               “plantName”: "example"}
             ],
     "fields": [
               {
               "name": “ListName”,
               "dataTypeID": 1043
               },
               {
               "name": "id_GatherList",
               "dataTypeID": 23
               },
               {
               "name": "idPlant",
               "dataTypeID": 23
               },
               {
               “Name”: “plantName”,
               “dataTypeID”: 1043
               }]}
```

1.1.1.6.11b getCollectedList

getCollectedList

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
```

```
},
"rowCount": {
  "type": "integer"
},
"oid": {
  "type": "integer"
},
"rows": {
  "type": "array",
  "items": [
    {
      "type": "object",
      "properties": {
        "PlantName": {
          "type": "string"
        },
        "idPlant": {
          "type": "string"
        }
      },
      "required": [
        "PlantName",
        "idPlant"
      ]
    }
  ]
},
"fields": {
  "type": "array",
  "items": [
    {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "dataTypeID": {
          "type": "integer"
        }
      },
      "required": [
        "name",
        "dataTypeID"
      ]
    },
    {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
```

```
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "SELECT",
     "rowCount": 1,
     "oid": 0,
     "rows": [ {
               "PlantName": "example",
               "idPlant": "example"
             } ],
     "fields": [
               {
               "name": “PlantName”,
               "dataTypeID": 1043
               },
               {
               "name": "idPlant",
               "dataTypeID": 23
               }]
}
```

1.1.1.6.12b removePlantFromCollectedList

removePlantFromCollectedList

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
```

```
"rowCount": {
  "type": "integer"
},
"oid": {
  "type": "integer"
},
"rows": {
  "type": "array",
  "items": [
    {
      "type": "object",
      "properties": {
        "idUser": {
          "type": "string"
        },
        "idPlant": {
          "type": "string"
        }
      },
      "required": [
        "idUser",
        "idPlant"
      ]
    }
  ]
},
"fields": {
  "type": "array",
  "items": [
    {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "dataTypeID": {
          "type": "integer"
        }
      },
      "required": [
        "name",
        "dataTypeID"
      ]
    },
    {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
```

```
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "DELETE",
     "rowCount": 1,
     "oid": 0,
     "rows": [ {
               "idUser": "example",
               "idPlant": "example"
             } ],
     "fields": [
               {
               "name": “idUser”,
               "dataTypeID": 23
               },
               {
               "name": "idPlant",
               "dataTypeID": 23
               }]
}
```

1.1.1.6.13b getPlantById

getPlantById

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
```

```
"rowCount": {
  "type": "integer"
},
"oid": {
  "type": "integer"
},
"rows": {
  "type": "array",
  "items": [
    {
      "type": "object",
      "properties": {
        "idPlant": {
          "type": "string"
        },
        "PlantName": {
          "type": "string"
        },
        "PlantPicture": {
          "type": "string"
        },
        "Edible": {
          "type": "string"
        },
        "PlantDescription": {
          "type": "string"
        },
        "PlantSteps": {
          "type": "string"
        },
        "Medicinal": {
          "type": "string"
        }
      },
      "required": [
        "idPlant",
        "PlantName",
        "PlantPicture",
        "Edible",
        "PlantDescription",
        "PlantSteps",
        "Medicinal"
      ]
    }
  ]
},
"fields": {
  "type": "array",
  "items": [
    {
```

```
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
```

```
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
      "name",
      "dataTypeID"
    ]
  },
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dataTypeID": {
        "type": "integer"
      }
    },
    "required": [
```

```
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

```
{
     "command": "SELECT",
     "rowCount": 1,
     "oid": 0,
     "rows": [ {
               "idPlant": "example",
               "PlantName": "example",
               “PlantPicture”: "example",
               “Edible”: "example",
               “PlantDescription”: "example",
               “PlantSteps”: "example",
               “Medicinal”: "example"
             } ],
     "fields": [
               {
               "name": “idPlant”,
               "dataTypeID": 23
               },
               {
               "name": "PlantName",
               "dataTypeID": 1043
               },
               {
               "name": "PlantPicture",
               "dataTypeID": 17
               },
               "name": "Edible",
               "dataTypeID": 21
               },
               {
               "name": "PlantDescription",
               "dataTypeID": 1043
               },
               {
```

```
                "name": "PlantSteps",
                "dataTypeID": 1043
                },
                {
                "name": "Medicinal",
                "dataTypeID": 21
                }]
}
```

1.1.1.6.14b getPlantsByRegion/1.1.1.6.15b getAllPlants

getPlantsByRegion/getAllPlants

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "command": {
      "type": "string"
    },
    "rowCount": {
      "type": "integer"
    },
    "oid": {
      "type": "integer"
    },
    "rows": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "idPlant": {
              "type": "string"
            },
            "plantName": {
              "type": "string"
            }
          },
          "required": [
            "idPlant",
            "plantName"
          ]
        }
      ]
    },
    "fields": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
```

```
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        },
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "dataTypeID": {
              "type": "integer"
            }
          },
          "required": [
            "name",
            "dataTypeID"
          ]
        }
      ]
    }
  },
  "required": [
    "command",
    "rowCount",
    "oid",
    "rows",
    "fields"
  ]
}
```

Example

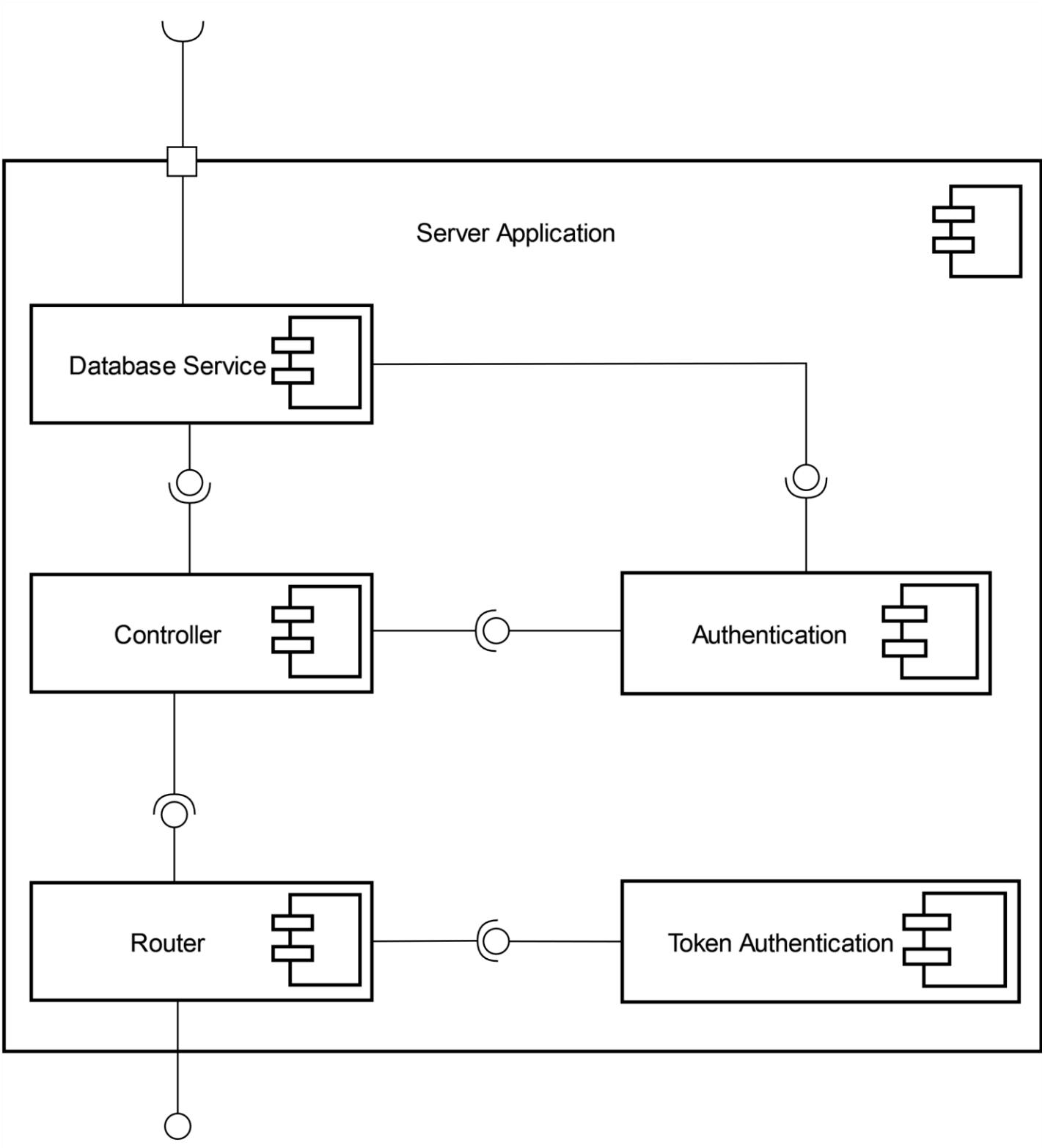
```
{
     "command": "SELECT",
     "rowCount": #,
     "oid": 0,
     "rows": [ {
               "idPlant": "example",
               "plantName": "example"
             },
             {...} ],
     "fields": [
               {
```

```
                "name": “idPlant”,
                "dataTypeID": 23
                },
                {
                "name": "plantName",
                "dataTypeID": 1043
                }]
}
```

Name	#1.1.1.6b Database Data JSON View
Purpose	Show the schema of what data is being returned for each query.
Description	This shows the properties of every schema returned by the queries.
Requirements	3, 6, 7, 10, 11, 14-20, 22, 23
Elements	**command:** Defines what type of query it is.
	rowCount: How many rows there are.
	oid: Object identifier; A system-level identifier used to uniquely identify various objects in the **Database**.
	rows: All the returned rows and their respective column values.
	fields: The columns in the row and their data types.
	1.1.1.6.1b getUserById: Retrieves all the information from a **User** by the specified id.
	1.1.1.6.2b getUserByUsername: Retrieves all the information from a **User** by the specified id.
	1.1.1.6.3b updatePassword: Updates the **User's** password.
	1.1.1.6.4b updateUsername: Updates the **User's** username.
	1.1.1.6.5b addGatherList: Inserts the new **Gather List's** name into the **Gather List** table, saves the last id from that table, then attaches that id with the specified user id in the User_has_Gather_List table.
	1.1.1.6.6b addPlantToGatherList: Inserts the id of a given **Plant** to a given **Gather List.**
	1.1.1.6.7b removePlantFromGatherList: Removes a given **Plant** from a given **Gather List.**
	1.1.1.6.8b addPlantToCollectedList: Adds a given **Plant's** id to the user's **Collected List.**
	1.1.1.6.9b getGatherListByUserId: Retrieves the name of the **Gather Lists** and their respective **Plant** information based on the given user id.
	1.1.1.6.10b getGatherListById: Retrieves the name of a **Gather List** and its respective **Plant** information based on the given **Gather List** id.

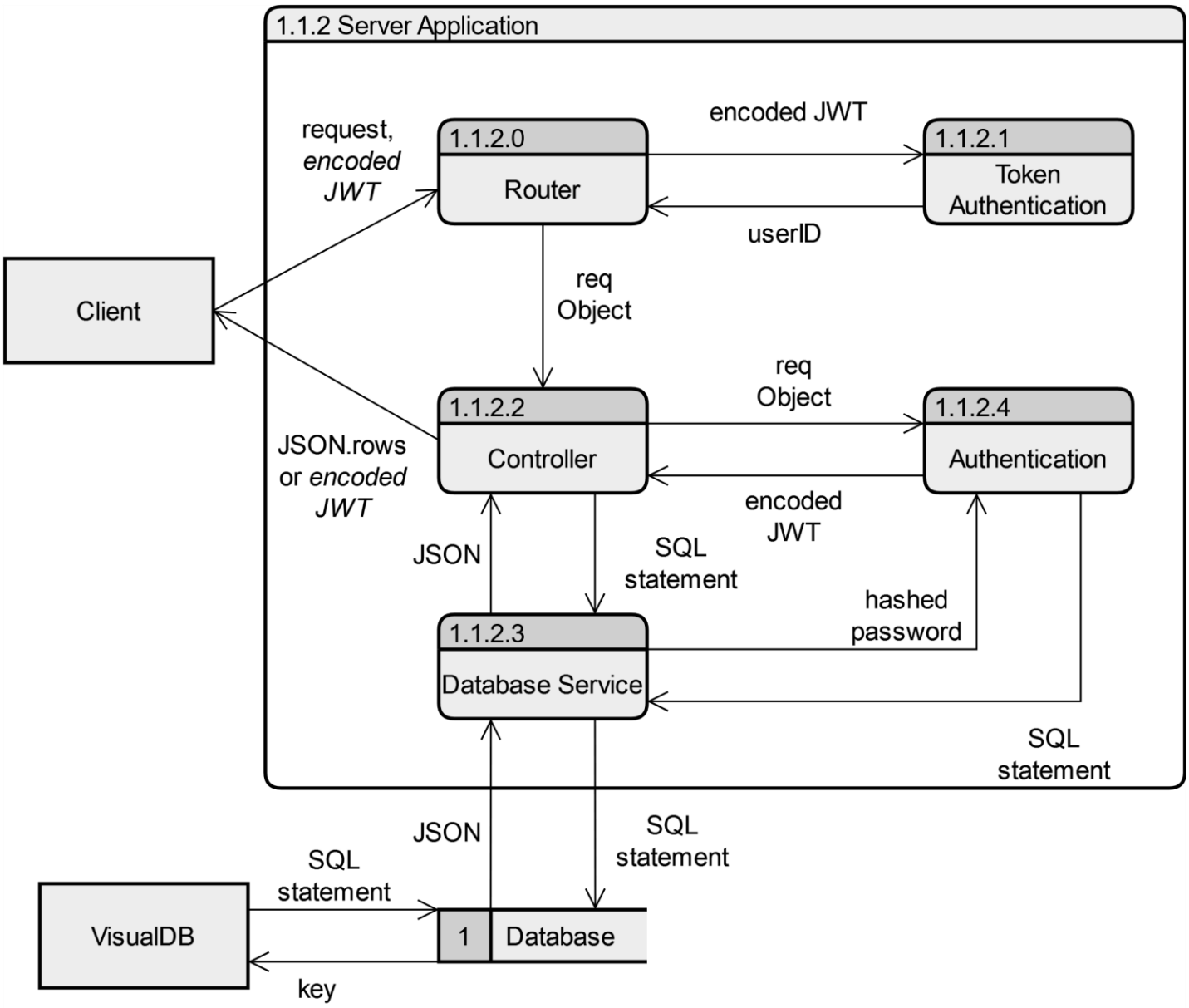
	1.1.1.6.11b getCollectedList: Retrieves the list of **Plants** that belong to the **Collected List** based on the given **User** id.
	1.1.1.6.12b removePlantFromCollectedList: Removes a given **Plant's** id based on the given **User** id.
	1.1.1.6.13b getPlantById: Retrieves the information from one **Plant** based on the given **Plant** id.
	1.1.1.6.14b getPlantsByRegion: Retrieves the information from all the **Plants** that belong to a given **Region** based on the provided **Region** name.
	1.1.1.6.15b getAllPlants: Retrieves the names and id's from every **Plant**.
Referenced By	**1.1.1.6 Database Queries View, 1.1 Server View, 1.1.1.3 Get Gather/Collected List**
Viewpoint	JSON Schema

View 1.1.2a: Server Application Component View



Name	#1.1.2a Server Application Component View
Purpose	A zoomed-in view of the **Server** Application.
Description	Shows the necessary components inside the **Server** Application in order to send or receive data from the **Database** and **Authenticate Users**.
Requirements	1, 4-7, 10, 14-21, 23
Elements	**Router:** Handles the requests from the **Client** and calls the appropriate functions from the controller.
	Controller: Contains all of the functions that are called to modify or retrieve data from the **Database**.
	Token Authorization: Logic for checking the **User's** existing web token.
	Authentication: Enables the **User** to log in or register.
	Database Service: A Node module (node-postgres) that allows querying of a database. (https://node-postgres.com/)
Referenced By	**1 Overall Application View, 1.1.2.5 User Sign Up View**
Viewpoint	Component Diagram

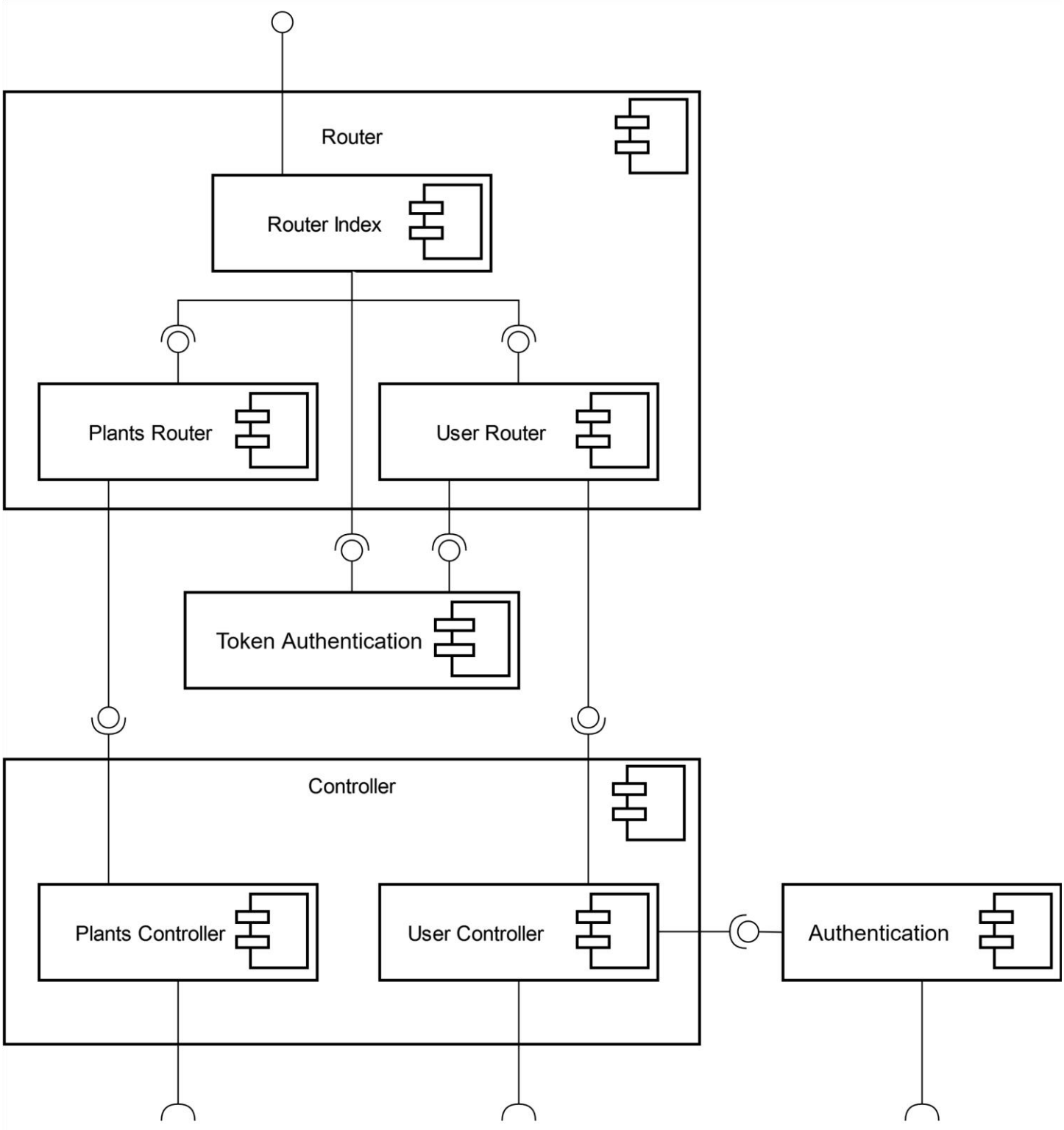
View 1.1.2b: Server Application View



Name	#1.1.2b Server Application View
Purpose	To show how the overall process of data flow occurs inside of the Server Application.
Description	The view shows how requests are sent from the **Client** to a router. This data then flows to the Controller which decides how to proceed with the request. It is sent to either **Database** Service directly, or Authentication depending on the request type. For most requests, a token received from the **Client** will be checked by Token Authentication prior to sending the request to the Controller.
Requirements	8, 9, 20-23
Elements	**1.2.1 Client**: the application on a mobile platform (Android).
	1.1.2 ServerApplication: the part of the server that responds to and processes requests from the **Client**.
	1.1.1 Database: the list of **User Credentials** as well as **Plant** data.
	1.1.2.0 Router: The receiver of requests and JWT from the **Client**.
	Encoded JWT: An encoded string used to authenticate the **User.** This is the form of the JWT in transit, and when sent to the **Client**.
	1.1.1 userID: The **User**'s Identification extracted from the Unencoded JWT Object in the form of an integer.
	Request: An HTTP message sent from the Client. (https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages)
	Req Object: An object that is made from the request that includes the userID as an attribute. (https://expressjs.com/en/api.html#req)
	1.1.1.6b JSON: A JSON file.
	1.1.1.6b JSON.rows: A subset of the JSON file, which is sent to **Client**.
	1.1.2.2 Controller: Returns responses to the **Client** and decides how requests should be handled.
	1.1.2.3 Database Service: Retrieves **Plant** data from the **Database**.
	1.1.2.1 Token Authentication: Confirms that the JWT is authentic.
	1.1.2.4 Authentication: A login/account creation process that interacts with the **Database** through editing or retrieving **User Credentials**. It verifies the login and creates a JWT.

	1.1.3 VisualDB: A 3rd party tool to edit the **Database**. Outside of the scope of this design. (https://visualdb.com/docs/)
Referenced By	**1 Overall Application View, 1.1 Server View**
Viewpoint	Data Flow Diagram

View 1.1.2.0a Router and Controller View



Name	#1.1.2.0a Router and Controller Component Diagram
Purpose	Provides an overview of the Router and Controller components, showing how they are divided between **Plant** and **User**.
Description	The index router file passes requests to either the **Plant** or **User** router files, which in turn pass the request to the appropriate controller function.
Requirements	1, 7, 10, 14-23
Elements	**Router Index:** The entry point of the **Server** application and the index file for the router.
	Plant Router: The **Plant** router file. Code in this file directs all endpoints on the /plant URI path to specific requests of the **Plant** controller.
	User Router: The **User** router file. Code in this file directs all endpoints on the /user URI path to specific requests of the **User** controller.
	Token Authentication: Logic for checking the **User's** existing web token. The Router Index calls this before accessing the **Plants** Router. The **User** Router requires this because it must selectively determine which of its endpoints require prior **Authentication**.
	Authentication: Authenticate the **User** when logging in or registering. The **User** Controller relies on this to complete these requests.
	Plant Controller: The **Plant** controller file. All **Database** operations relating to **Plants** are executed from here.
	User Controller: The **User** controller file. All **Database** operations relating to **User** data, including **Credentials** and list data, are executed from here.
Referenced By	**1.1.2 Server Application**
Viewpoint	Component Diagram

View 1.1.2.0b Router Pseudocode View

1.1.2.0b.1 accountLogin

```
router.GET(“/login”, authentication.accountLogin)
```

1.1.2.0b.2 accountRegistration

```
router.POST(“/registration”, authentication.accountRegistration)
```

1.1.2.0b.3 getUserById

```
router.GET(“/user”, userController.getUserById)
```

1.1.2.0b.4 getUserByUsername

```
router.GET(“/user/:username”, userController.getUserByUsername)
```

1.1.2.0b.5 updatePassword

```
router.PATCH(“/user/password”, userController.updatePassword)
```

1.1.2.0b.6 updateUsername

```
router.PATCH(“/user/username”, userController.updateUsername)
```

1.1.2.0b.7 addGatherList

```
router.POST(“/gather”, userController.addGatherList)
```

1.1.2.0b.8 addPlantToGatherList

```
router.POST(“/gather/:plantId”, userController.addPlantToGatherList)
```

1.1.2.0b.9 removePlantFromGatherList

```
router.DELETE(“/gather/:plantId”, userController.removePlantFromGatherList)
```

1.1.2.0b.10 getGatherListByUserId

```
router.GET(“/gather”, userController.getGatherListByUserId)
```

1.1.2.0b.11 getGatherListById

```
router.GET(“/gather/:id”, userController.getGatherListById)
```

1.1.2.0b.12 addPlantToCollectedList

```
router.POST(“/collected/:plantId”, userController.addPlantToCollectedList)
```

1.1.2.0b.13 getCollectedList

```
router.GET(“/collected”, userController.getCollectedList)
```

1.1.2.0b.14 removePlantFromCollectedList

```
router.DELETE(“/collected/:plantId”, userController.removePlantFromCollectedList)
```

1.1.2.0b.15 getAllPlants

```
router.GET(“/plant”, plantController.getAllPlants)
```

1.1.2.0b.16 getPlantById

```
router.GET(“/plant/:id”, plantController.getPlantById)
```

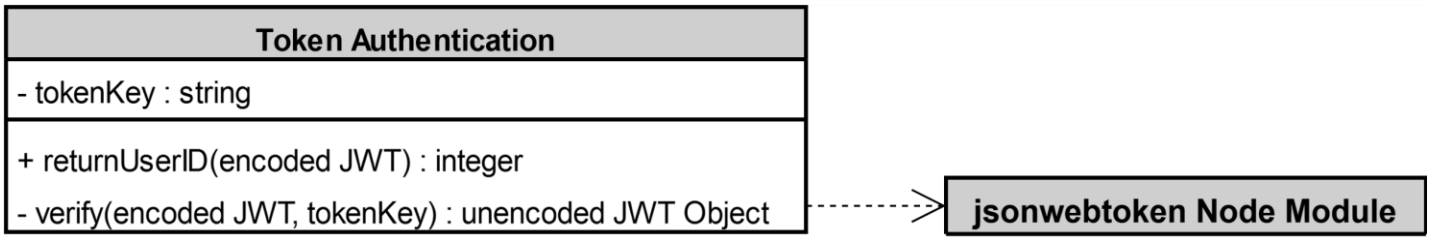
1.1.2.0b.17 getLocalPlants

```
router.GET(“/localPlants:coordinates”, plantController.getLocalPlants)
```

Name	#1.1.2.0b Router Pseudocode View
Purpose	Show all of the routes the server will utilize.
Description	There are various routes that allow the user to insert and retrieve data according to the HTTP request method and URL path. Each one connects to a specific controller function.
Requirements	3, 6, 7, 10, 11, 13, 14-19
Elements	**1.1.2.0b.1 accountLogin:** Logs in the **User**.
	1.1.2.0b.2 accountRegistration: Registers the **User**.
	1.1.2.0b.3 getUserById: Retrieves all the information from the logged in **User.**
	1.1.2.0b.4 getUserByUsername: Retrieves all the information from a **User** by the specified username.
	1.1.2.0b.5 updatePassword: Updates the logged in **User's** password.
	1.1.2.0b.6 updateUsername: Updates the logged in **User's** username.
	1.1.2.0b.7 addGatherList: Inserts the new **Gather List's** name into the **Gather List** table, saves the last id from that table, then attaches that id with the logged in **User** id in the User_has_Gather_List table.
	1.1.2.0b.8 addPlantToGatherList: Inserts the id of a given **Plant** to a logged in **User's Gather List.**
	1.1.2.0b.9 removePlantFromGatherList: Removes a given **Plant** from a logged in **User's Gather List.**
	1.1.2.0b.10 getGatherListByUserId: Retrieves the name of the **Gather Lists** and their respective **Plant** information based on the logged in **User** id.
	1.1.2.0b.11 getGatherListById: Retrieves the name of a **Gather List** and its respective **Plant** information based on the logged in **Gather List** id.
	1.1.2.0b.12 addPlantToCollectedList: Adds a given **Plant's** id to the **User's Collected List.**
	1.1.2.0b.13 getCollectedList: Retrieves the list of **Plants** that belong to the **Collected List** based on the logged in **User** id.
	1.1.2.0b.14 removePlantFromCollectedList: Removes a given **Plant's** id based on the logged in **User** id.
	1.1.2.0b.15 getAllPlants: Retrieves the name and id of every **Plant**.

	1.1.2.0b.16 getPlantById: Retrieves the information from one **Plant** based on the given **Plant** id.
	1.1.2.0b.17 getLocalPlants: Retrieves the information from all the **Plants** that are located within a certain distance of the **User**.
	GET: HTTP method for retrieving data.
	POST: HTTP method inserting data.
	PATCH: HTTP method for updating data.
	DELETE: HTTP method for deleting data.
	1.1.1.6a userController: File that contains all of the functions that perform queries regarding the **User's** information.
	1.1.1.6a plantController: File that contains all of the functions that perform queries regarding **Plant** information.
Referenced By	
Viewpoint	Pseudocode

View 1.1.2.1a: Token Authentication Class View



Name	#1.1.2.1a Token Authentication Class View
Purpose	This view shows what each function returns and shows that the Token Authentication holds the private key.
Description	The Token Authentication holds a private key that is used to verify the encoded JWT. Verify then returns an unencoded JWT object which is used to retrieve the userID from the JWT. It sends this userID back to the caller.
Requirements	20, 22
Elements	**returnUserID**: Returns the userID to the caller after it receives an unencoded JWT Object from the Verify function
	1.1.1 userID: The **User**'s Identification extracted from the Unencoded JWT Object in the form of an integer.
	Verify: A function from the jsonwebtoken Node module used to verify that the token provided is authentic and will return an unencoded JWT Object. It is outside the scope of this project. (https://www.npmjs.com/package/jsonwebtoken#jwtverifytoken-secretorpublickey-options-callback)
	jsonwebtoken Node Module: A library that consists of functions that create and interact with JSON Web Tokens.
	1.1.2.3 Encoded JWT: An encoded string used to authenticate the **User.** This is the form of the JWT in transit, and when sent to the **Client**.
	1.1.2.3 Unencoded JWT Object: The full contents of the JWT in its unencoded form.
Referenced By	**1.1.2b Server Application View, 1.1.1.1 Update Gather List View, 1.1.1.2 Checking Off From Gather List View, 1.1.1.3 Get Gather/Collected List View, 1.1.2b Server Application View**
Viewpoint	Class Diagram

View 1.1.2.1b Token Authentication Psuedocode View

```
returnUserID(encodedJWT)
     unencodedJWT ← jsonwebtokenNodeModule.verify(encodedJWT, tokenKey)
     RETURN unencodedJWT.userID
```

Name	#1.1.2.1b Token Authentication Pseudocode View
Purpose	This view shows how we retrieve a userID from the Encoded JWT state.
Description	The Token Authentication holds a private key that is used to verify the encoded JWT. Verify then returns an unencoded JWT object which is used to retrieve the userID from the JWT. It sends this userID back to the caller.
Requirements	20, 22
Elements	**returnUserID**: Takes an encoded JWT and calls verify with it and the key. Returns the userID to the caller after it receives an unencoded JWT Object from the Verify function.
	1.1.1 userID: The **User**'s Identification extracted from the Unencoded JWT Object in the form of an integer.
	Verify: A function from the jsonwebtoken Node module used to verify that the token provided is authentic and will return an unencoded JWT Object. It is outside the scope of this project. (https://www.npmjs.com/package/jsonwebtoken#jwtverifytoken-secretorpublickey-options-callback)
	jsonwebtoken Node Module: A library that consists of functions that create and interact with JSON Web Tokens.
	1.1.2.3 Encoded JWT: An encoded string used to authenticate the **User.** This is the form of the JWT in transit, and when sent to the **Client**.
	1.1.2.3 Unencoded JWT Object: The full contents of the JWT in its unencoded form.
Referenced By	**1.1.2b Server Application View, 1.1.1.1 Update Gather List View, 1.1.1.2 Checking Off From Gather List View, 1.1.1.3 Get Gather/Collected List View, 1.1.2b Server Application View**
Viewpoint	Pseudocode

View 1.1.2.2 Controller Pseudocode View

```
adjust(point,by)
     adjusted ← …point
     adjusted.latitude increment by[0]
     adjusted.longitude increment by[1]
     Return(adjusted.latitude, adjusted.longitude)


getRegionFromGPS(userGPS)
     // Using the circle function we can find the 4 cardinal
         directions Lat/Long 10km away on a curved surface.

     outerbounds ← circle(userGPS, radius:10).calculateOuterbounds
     bounds ← {
          north ← adjust(userGPS, [outerbounds.deltaLatitude,0])
          south ← adjust(userGPS, [-outerbounds.deltaLatitude,0])
          east ← adjust(userGPS, [0,outerbounds.deltaLongitude])
          west ← adjust(userGPS, [0,-outerbounds.deltaLongitude])
     }

     // We are rounding the coordinates to find out the user's
         region. Region IDs are created the same way with plants in
         the database.

     FOR direction in bounds
          direction ← (roundup(direction[0]),roundup(direction[1])

     // Create a List of the range of possible regions inside the
         longitudes of the southern and northern ranges

     FOR num in range(south[1], north[1] + 1, 1)
          regionLongitudeFromGPS ← num

     FOR num in range(west[0], east[0] + 1, 1)
```

```
          regionLatitudeFromGPS ← num

     // Create a list of all regions within the box of cardinal bounds.

     FOR coord1 in regionLatitudeFromGPS
          FOR coord2 in regionLongitudeFromGPS
               regionIDListFromGPS ← "regionLatitudeFromGPS[coord1],
               regionLongitudeFromGPS[coord2]"

     RETURN regionIDListFromGPS

GetLocalPlants(userPosition, GPS)
     regionIDList <- getRegionFromGPS(GPS)

     // Get a list of plants present in the given regions
     FOR regionID in regionIDList
          plantIDsFoundList <- getUniquePlantIDsByRegion(regionID)

      // Check each region within the searchRadius to see if a given plant is
      within the searchRadius
      FOR regionID in regionIDList
          FOR plantID in plantIDsFoundList
               sightingPositionsInRegion <-
               getSightingsByPlantAndRegion(plantID, regionID)

               SightingInRange(sightingsInRegion, localPlants)
     RETURN localPlants

SightingInRange(sightingPositionsInRegion, localPlants)
     FOR sightingPosition in sightingsPositionsInRegion
          IF InRange(userPosition, sightingPosition, searchRadius)
               localPlants.Add(plantID)
               RETURN
```

```
InRange(userPosition, sightingPosition, searchRadius)
     distanceToPlant <-
     geolib.getDistance(userPosition, sightingPosition)

     IF distanceToPlant < searchRadius
          RETURN true
     RETURN false
```

Name	#1.1.2.2 Controller Pseudocode View
Purpose	To gather the list of regions that intersect our **User**'s radius so that we can narrow down the list of sightings to look through to send back to **User** as a **Local Plants** list. The returns a list of **Local Plants**.
Description	These functions will find the north, south, east, and west Latitude and Longitude 10km away from the **User** that accounts for the curvature of the Earth. Using these, it will create a list of regionID's that are within these bounds. Then, the GetLocalPlants function will query the **Database** for a list of sightings within the regions that overlap with the **User's** search radius. It will then check which plants are within the searchRadius and return that list.
Requirements	3, 4
Elements	**1.1.1 IdRegion:** Generated from rounding up the longitude and latitude and combining them in a string format.
	Circle: A function designed by Roger Martin with the purpose of finding the cardinal directions' GPS **Location** bound with the parameters of the **User**'s **Location** and radius in KM that accounts the curvature of the Earth. Roger Martin: (https://math.stackexchange.com/users/1009736/roger-martin), Reverse use of Haversine formula, URL (version: 2021-12-28): https://math.stackexchange.com/q/4343818
	Adjust: An adjusting function written by Roger Martin that edits the **Location** returned from the Circle function. Roger Martin: (https://math.stackexchange.com/users/1009736/roger-martin), Reverse use of Haversine formula, URL (version: 2021-12-28): https://math.stackexchange.com/q/4343818
	Bounds: The list of **Locations** that correlate to the cardinal directions.
	UserGPS: The **User**'s GPS location in a tuple (latitude, longitude).
	regionLatitudeFromGPS: A list of latitude points used to concatenate with the items in regionLongitudeFromGPS to create regionIDs.
	regionLongitudeFromGPS: A list of longitude points used to concatenate with the items in regionLatitudeFromGPS to create regionIDs.
	RegionIDListFromGPS: A list of all regionID's that are within the **User**'s radius of 10km.

	inRange: Returns true if a sighting position is within the **User's** searchRadius
	sightingInRange: Check if a given sighting is within the searchRadius, if so add to nearbyPlants and stop searching
	getLocalPlants: gets a list of **Local Plants**
	1.1.1.6.13a getUniquePlantIDsByRegion: Retrieves all unique **Plant** ids from Sighting table by **Region**.
	1.1.1.6.16a getSightingsByPlantAndRegion: Get all **Locations** from Sighting table, for a specific **Plant** in a specific **Region**.
	GetRegionListFromGPS: Function that will find the north, south, east, and west latitude and longitude 10km away from the **User** that accounts for the curvature of the Earth. Using these, it will create a list of region ID's that are within these bounds.
	geolib.getDistance: An external library to get the distance between **Locations**. (https://www.npmjs.com/package/geolib)
Referenced By	**1.1.2b Server Application View, 1.1.2.0a Router and Controller View**
Viewpoint	Pseudocode

View 1.1.2.3 JSON Web Token View

Unencoded JWT Object

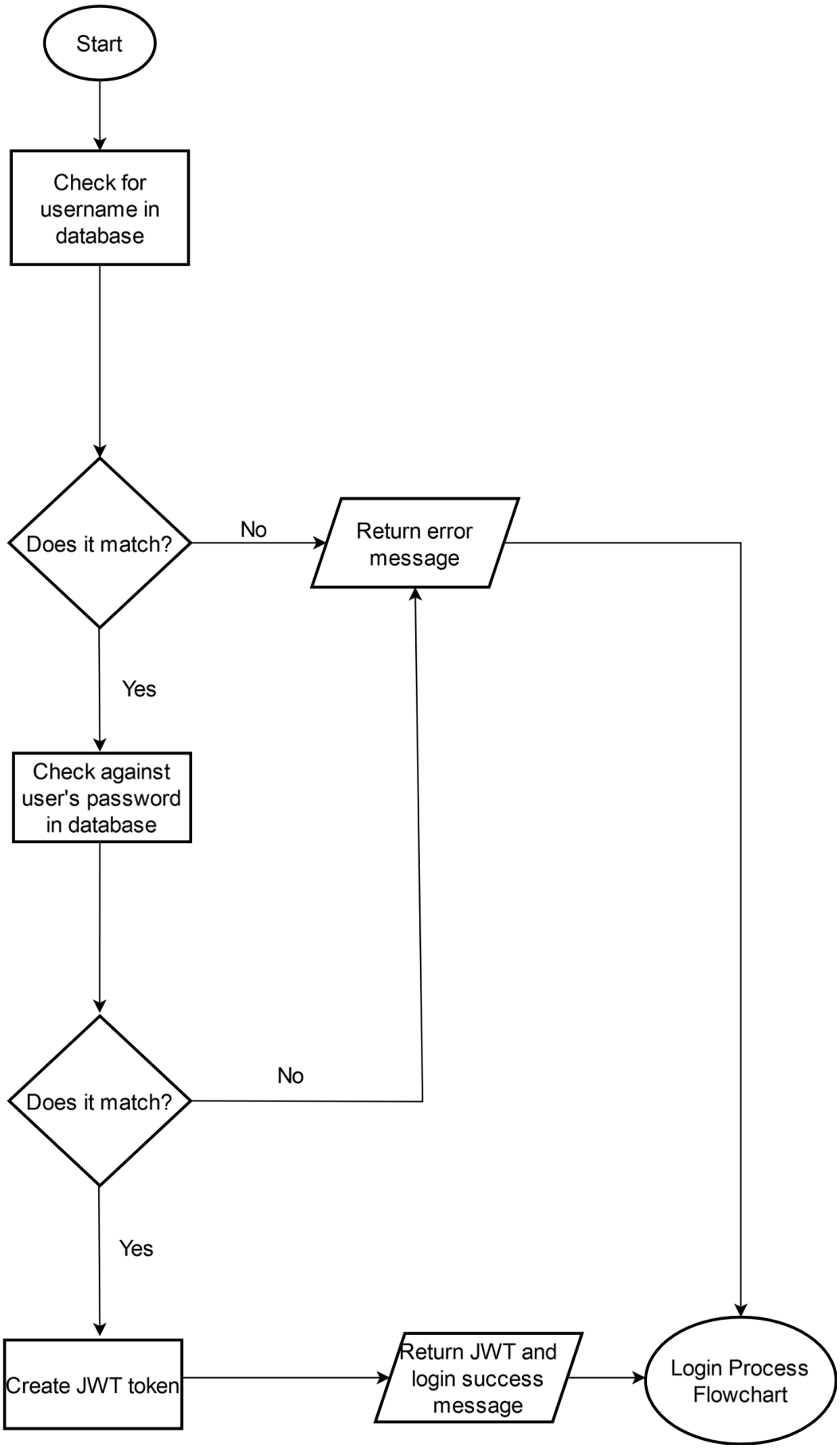

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "header": {
      "type": "object",
      "properties": {
        "alg": {
          "type": "string"
        },
        "typ": {
          "type": "string"
        }
      },
      "required": [
        "alg",
        "typ"
      ]
    },
    "payload": {
      "type": "object",
      "properties": {
        "userID": {
          "type": "integer"
        }
      },
      "required": [
        "userID"
      ]
    },
    "signature": {
      "type": "string"
    }
  },
  "required": [
    "header",
    "payload",
    "signature"
  ]
}
```


Encoded JWT


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4g
RG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Name	#1.1.2.3 JSON Web Token View
Purpose	Show the contents and format of the JSON Web Token (JWT).
Description	The unencoded JWT is in JSON format, containing a header, payload, and signature. The encoded JWT stores the same information as a Base64Url-encoded string.
Requirements	19-21
Elements	**Unencoded JWT Object:** The full contents of the JWT in its unencoded form.
	header: Describes metadata about the token, including alg and typ.
	alg: Specifies the algorithm used to sign the JWT. Here, "HS256" indicates HMAC SHA-256.
	typ: Defines the token type as “JWT” to specify this as a JSON Web Token, as is standard convention.
	payload: Contains all data being transferred, in this case the userID.
	1.1.1 userID: The id of the **user**, matching the id stored in the **Database** when the JWT was created.
	signature: Used to verify the token’s authenticity. Created using the **Server’s** secret key combined with the Base64-encoded contents of the header and payload, and hashed using the algorithm specified in the header.
	Encoded JWT: An encoded string used to authenticate the **User**. This is the form of the JWT in transit, and when sent to the **Client**.
Referenced By	**1.1.1.1 Update Gather List View, 1.1.1.2 Checking Off from Gather List View, 1.1.1.3 Get Gather/Collected List View, 1.1.2b Server Application View, 1.1.2.1 Token Authentication View, 1.1.2.4a Login Server Flowchart View, 1.1.2.5 User Sign Up View, 1.2.1.1a Login Process Flowchart View, 1.2.1.1b User Login Process View, 1.2.1.1c Authentication Token from Login, 1.2.3.4a Send Authentication Token, 1.2.3.4b Removal of Authentication Token**
Viewpoint	JSON Schema

View 1.1.2.4a: Login Authe Flowchart View



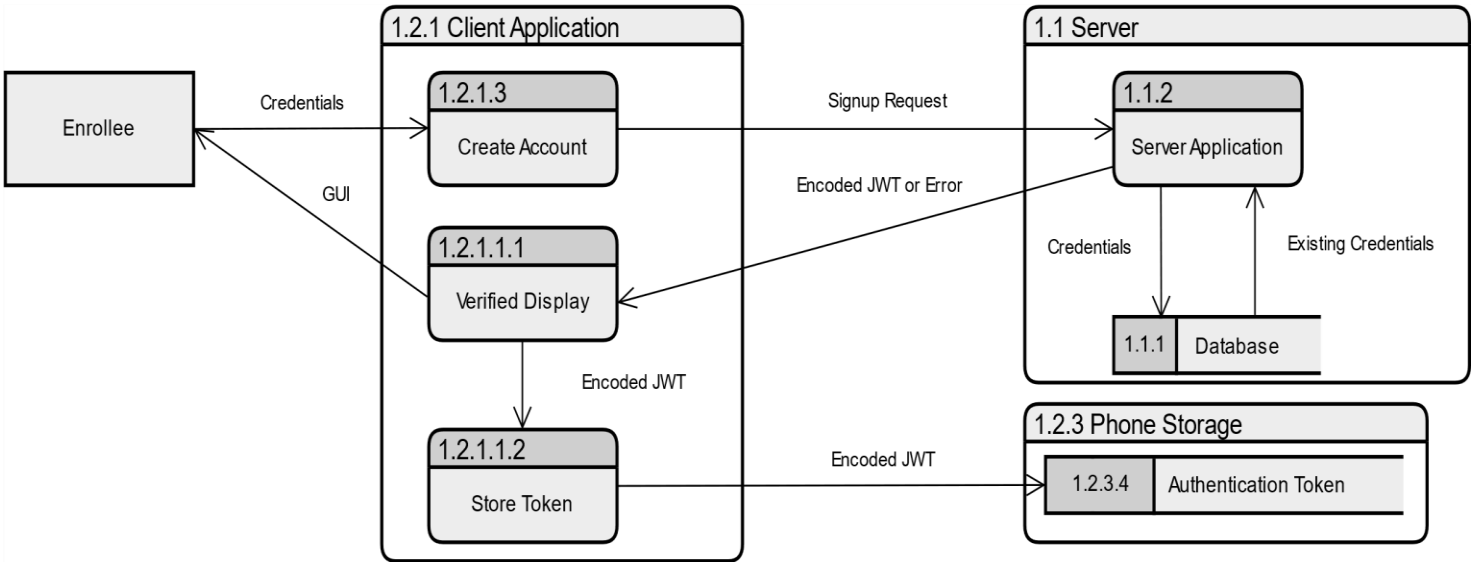
Name	#1.1.2.4a Login Server Flowchart
Purpose	Shows the decision-making process on the **Server** when a **User** logs in.
Description	When a **Requester** makes a login request, the **Authentication** service will check the given **Credentials** against those stored in the **Database**. If the **Credentials** match, the service will generate a JWT, return it out of the scope, and log the **Requester** in as a **User**.
Requirements	20, 23
Elements	**1.1.1 Database:** A collection of plant information and **User** credentials.
	1.1.2.3 JWT (JSON Web Token)**:** Used to confirm prior user authentication.
	1.2.1.1a Login Process Flowchart: Details the process of logging into the **App**.
	1.1.2.4 Authentication: A login/account creation process that interacts with the database through editing or retrieving user credentials. It verifies the login and creates a JWT.
Referenced By	**1.1.2a Server Application View**
Viewpoint	Flowchart

View 1.1.2.4b: Authentication View

```
accountLogin(req, res)
     accountUsername, accountPassword -> req.body
     accountData -> getAccountByUsername(accountUsername)
     IF accountData not valid
          PUT error message
          RENDER login page
          RETURN
     TRY
          IF accountPassword = accountData.password
               token -> jwt.sign(accountData)
               res.cookie(token)
          REDIRECT to account page
     CATCH(error)
          RETURN error

accountRegistration(req, res)
     accountUsername, accountPassword, accountEmail -> req.body
      accountData -> getUserByUsername(accountUsername)
      IF NOT accountData
            TRY
                 hashedPassword = bcrypt.hashSync(accountPassword, 10)
                 registerUser(accountUsername, hashedPassword, accountEmail)
            CATCH(error)
                  RETURN error

registerUser(username, password, email)
     data -> database.query(“INSERT INTO User (Username, HashedPassword,
RecoveryEmail) VALUES ({username}, {password}, {email})”)
```

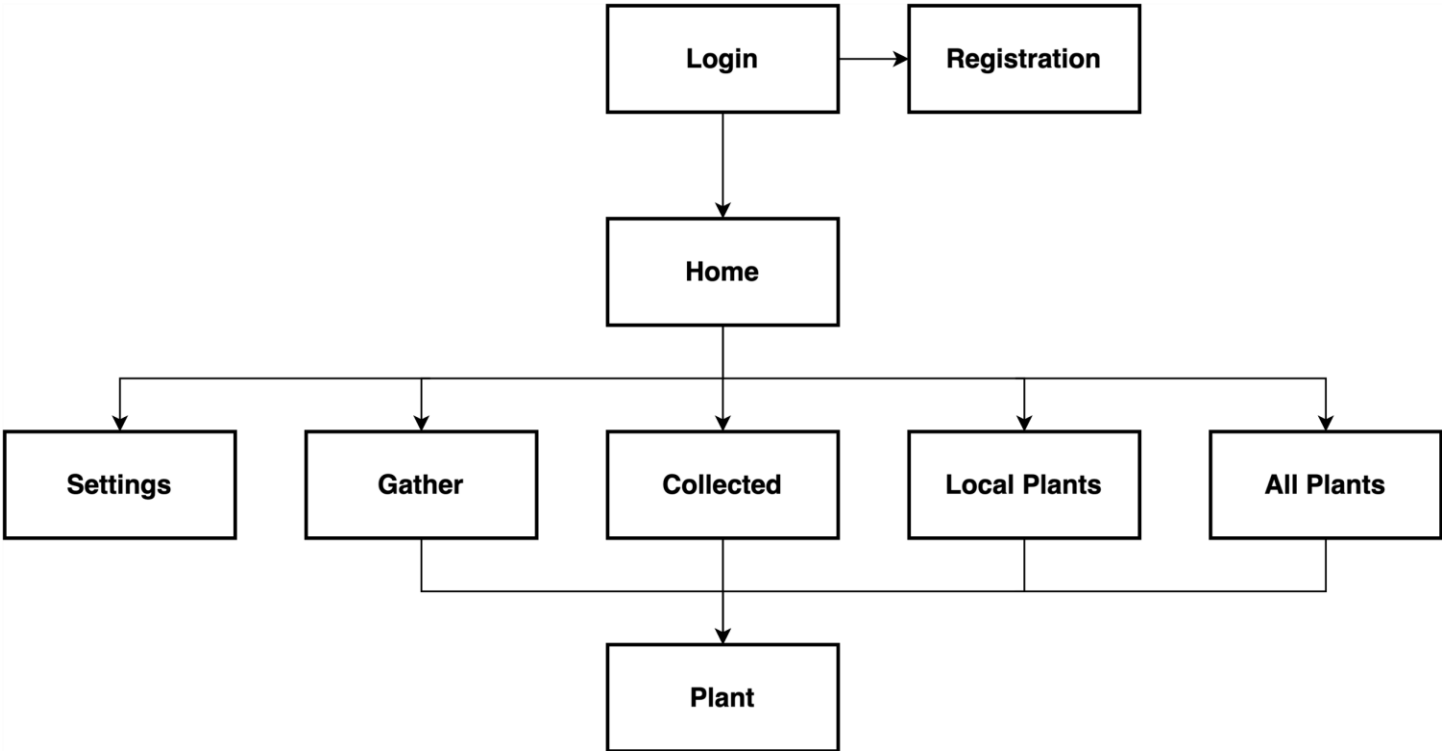
Name	# 1.1.2.4b Authentication View
Purpose	Show the logic behind the login process
Description	The function calls another function which queries the **User's Credentials** based on their username, then compares the **User's** password with the one provided at login.
Requirements	21, 22
Elements	**1.1.1.6.15a getAccountByUsername**: Function that queries the **User's Credentials** by the **Username** they entered at login.
	registerUser: Function that creates a new **User** in the **Database** with the given **Credentials**.
	result: The returned rows of data from the query.
	accountLogin: Function for handling the login process.
	req: The HTTP request sent from the **Client** to the **Server**.
	res: The response from the **Server** back to the **Client**.
	accountUsername: Username that the **User** entered at login.
	accountPassword: Password that the **User** entered at login.
	accountData: The returned value from getAccountByUsername.
	bcrypt: An external library used to handle password hashing. (https://www.npmjs.com/package/bcrypt)
	jwt: JSON Web Token library which holds functions to create and authenticate tokens. (https://www.npmjs.com/package/jsonwebtoken)
	cookie: Stores the encoded JWT object.
Referenced By	**1.1.2.4a Login Server Flowchart**
Viewpoint	Pseudocode

View 1.1.2.5 User Sign Up View



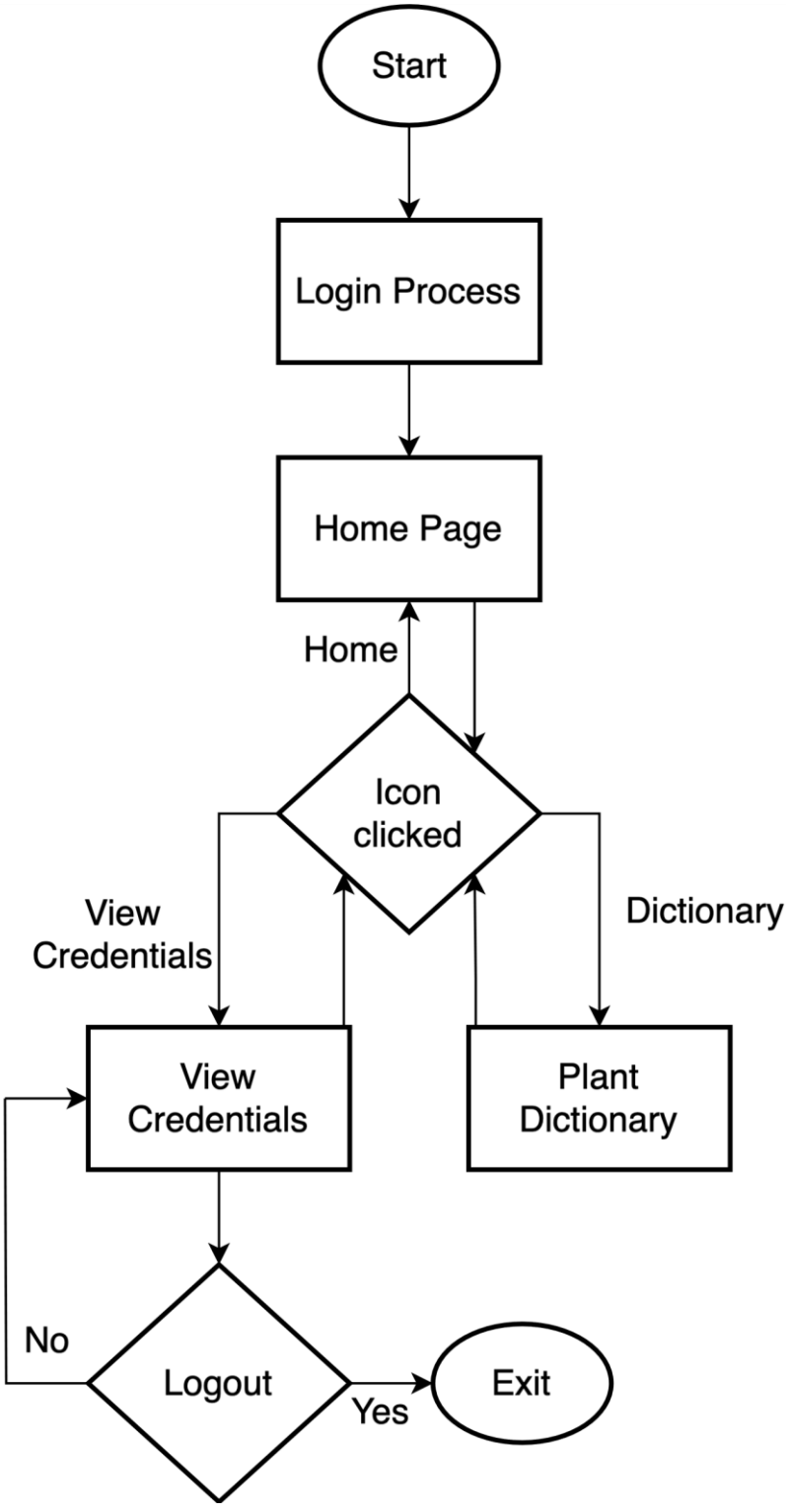
Name	#1.1.2.5 User Sign Up View
Purpose	This shows the process of an **Enrollee** creating an account and subsequently getting **Authenticated** and becoming an **User**.
Description	An **Enrollee** will provide **Credentials** to create an account in the **App**. The **App** will validate the **Credentials**, and send a request to create an account to the **Server**. Once the **Server** validates the request, a JSON Web Token (JWT) will be sent back to the **App**. The **App** will then store the JWT in the phone's storage system.
Requirements	19, 21, 22
Elements	**1.2.1.3 Create Account:** The **Enrollee** interacts with the **App** and provides it with **Credentials** to create an account.
	1.2.1.1.1 Verified Display: Displays to the **Enrollee** if the account creation was successful or not.
	1.2.1.1.2 Store Token: The **App** will receive the new authentication JSON and send it to the phone's storage system.
	Signup Request: The **Enrollee's** username and password as strings, encrypted and then packaged and sent as an http request.
	Authentication Token: The Encoded JWT that is stored in the phone's storage system.
	1.1 Server: The back-end part of the **Client**-**Server** pair.
	1.1.2 Server Application: Receives requests from the client application, decrypts, parses, and compares the requests with the database, sending back either a validated Authentication JSON or an error.
	1.1.1 Database: Backend storage of the server.
	1.1.2.2 Encoded JWT: An encoded string used to authenticate the **User**. This is the form of the JWT in transit, and when sent to the **Client**.
	Error: A response from the server containing relevant information to be shown to the **Enrollee**, prompting them to try again.
Referenced By	**1.2.1.1 Login Process View, 1.2.1.3 Create Account View**
Viewpoint	Data Flow Diagram

View 1.2.1a: Client Application Site Map View



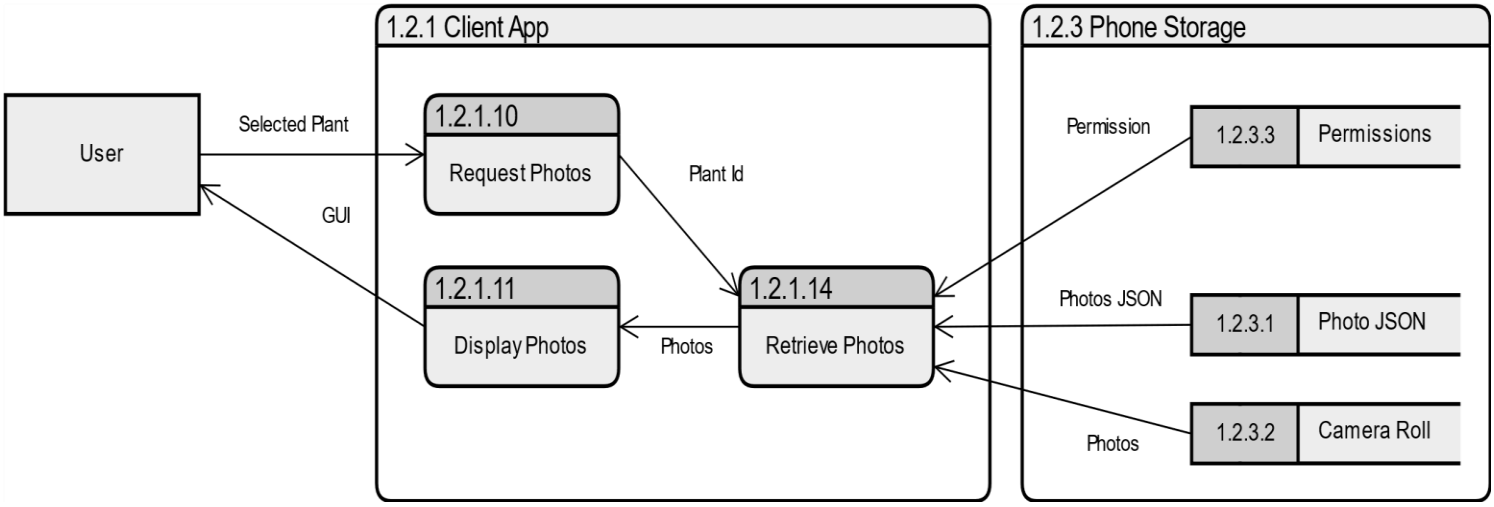
Site Map	#1.2.1a Site Map View
Purpose	To give a layout of all the pages in the **App** and how they navigate.
Description	The Site Map shows the overall layout and navigation of the **App**. Basic page information is included on the **Plant** page.
Requirements	3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16 ,17, 18, 19, 20, 21
Elements	**Login:** The page that authenticates the **User**.
	Registration: The page that registers a new **User**.
	Home: The page that contains the main dashboard and navigation to the rest of the **App**.
	Settings: The page that allows the **User** to toggle permissions or update their account information.
	Gather: The page containing one of the **User's Gather Lists**.
	Collected: The page containing the **User's Tick List**.
	Local Plants: The page containing the **User's Local Plants** L**ist**.
	All Plants: The page containing the **All Plants List**.
	Plant: The page containing the **Photos**, the **Region**, **Related Plant**, **Use Type**, **Name**, **Description**, and **Steps** for a **Plant**.
Referenced By	**1 Overall Application View**
Viewpoint	Site Map

View 1.2.1b: Overall User Flow Diagram View



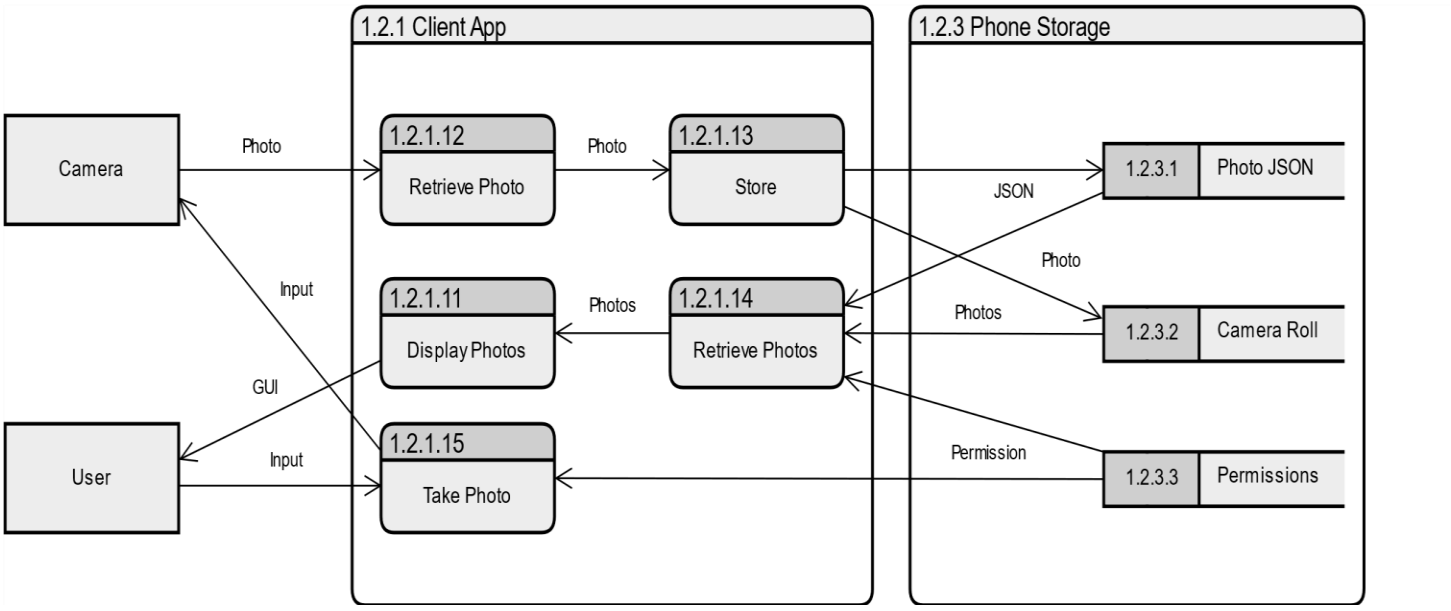
Name	View 1.2.1b Overall User Flow Diagram View
Purpose	Shows the flow of the **App** from the **User's** perspective.
Description	The **App** will start with the login process before allowing access to the **App**. From there they can access the home page, plant dictionary, or view credentials from the icons on the **App**. In the view credentials, the **User** is capable of logging out of the **App**.
Requirements	10, 19, 20
Elements	**1.2.1.1 Login Process:** The first screen of the **App** which authenticates the **User** by prompting them for their **Credentials**.
	1.2.1.4 Home page: The main screen for the **App**.
	Plant Dictionary: A list of all the **Plants** in the **Database**.
	1.2.1.16 View Credentials: Where the **User's** account information can be viewed, and **User** is able to log out.
	1.2.1.16.1 Logout: A decision where the **User** will be asked "Are you sure you wish to logout?" The **User** will answer "Yes" or "No". Then the **User** is either logged out of the **App** and its services or they're returned to view credentials.
Referenced By	**1 Overall Application View, 1.2.1.1a Login Process View, 1.2.1.7 Find Plants Flowchart View**
Viewpoint	Flowchart

View 1.2.1c: User Photo Data Flow Diagram



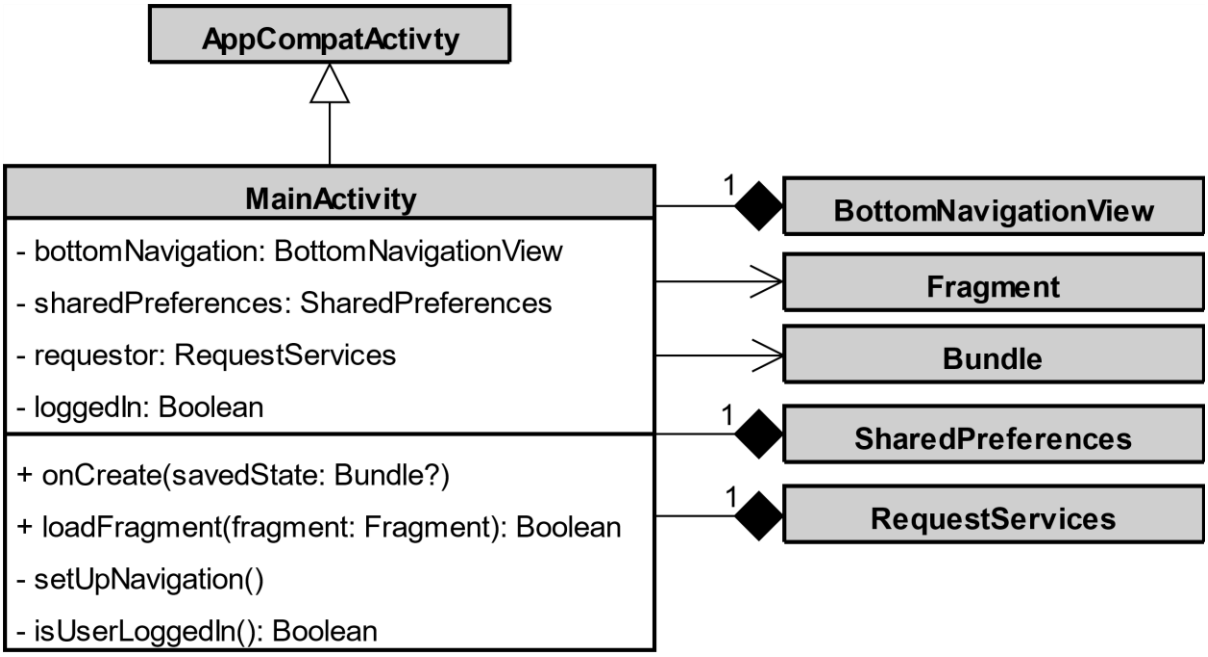
Name	#1.2.1c User Photo Data Flow Diagram
Purpose	Illustrates the flow of data when a **User** wants to retrieve a **Photo** that is attached to a specific **Plant**.
Description	**App** accesses the **Photos JSON**, retrieves the **Photo** data, retrieves the **Photo** from the camera roll, and displays it to the **User**.
Requirements	7, 13
Elements	**1.2.1.10 Request Photos:** The **User** requests **Photos** relating to a selected **Plant**.
	1.2.1.14 Retrieve Photos: The **App** retrieves the **Photos** associated with the specified **Plant** from the camera roll.
	1.2.3.3 Permissions: Device permissions for the **Camera**, **Camera Roll**, and **GPS.**
	1.2.3.1 Photos JSON: A collection of **Plant** and **Photo** data as key-value pairs that is stored in local storage.
	1.2.3.2 Camera Roll: A collection of locally stored **Photos**, accessed by the **App** through phone services once permission has been granted.
	Display Photos: The **App** displays the **Photos** to the **User**.
	Photos: Pictures will be passed in a list. The list will include filenames for each **Photo**.
	1.2.3 Phone Storage: Handles the overall file system on the **User's** device.
	Plant Id: Each **Plant** has an integer Id assigned by the **Database**.
Referenced By	**1.2.2 Phone Services, 1.2.3 Phone Storage, 1.1.1 Database View**
Viewpoint	Data Flow Diagram

View 1.2.1d: User Takes Photo Data Flow Diagram



Name	#1.2.1d User Photo Data Flow Diagram
Purpose	Illustrates the process and flow of data when a **User** captures a **Photo** of a specific **Plant** using the **App,** links the **Photo** to the **Plant,** and saves it to local storage.
Description	**App** accesses the device's camera through **Phone Services**, captures the **Photo**, stores the **Plant** and **Photo** as a key-value pair in local storage, stores the **Photo** in the camera roll, and then displays it to the **User.**
Requirements	12, 13
Elements	**Take Photo:** The **User** sends a request to the device's camera to initiate access, once permission is granted, the **User** can take a **Photo**.
	Camera: The device's built in camera, accessed by the **App** through phone services**.** Can only be used if the **User** has granted permission**.**
	Retrieve Photo: After the **User** has captured an image using the camera, the **App** retrieves the **Photo** from the camera.
	Store: The **App** sends the captured image and **Photo** data to the device's local storage**.**
	1.2.3 Phone Storage: Handles the overall file system on the **User's** device.
	1.2.3.1 Photos JSON: A collection of **Plant** and **Photo** data key-value pairs that is stored in local storage.
	1.2.3.2 Camera Roll: The device's camera roll. Can only be used if the **User** has granted permission**.**
	1.2.1.14 Retrieve Photos: The **App** retrieves the **Photo** data from the **Phone Key** and the **Photo** from the camera roll. If the **User** has granted permission.
	Display Photos: The **App** displays the **Photo** to the **User.**
	1.2.3.3 Permissions: Device permissions for the **Camera**, **Camera Roll**, and **GPS.**
	Photo: The actual photo being passed. PNGs are the preferred format for the **Plant** images.
	Photos: Pictures will be passed in a list. The list will include filenames for each Photo.
Referenced By	**1.2.2 Phone Services, 1.2.3 Phone Storage**
Viewpoint	Data Flow Diagram

View 1.2.1e: Main Activity Class Diagram View



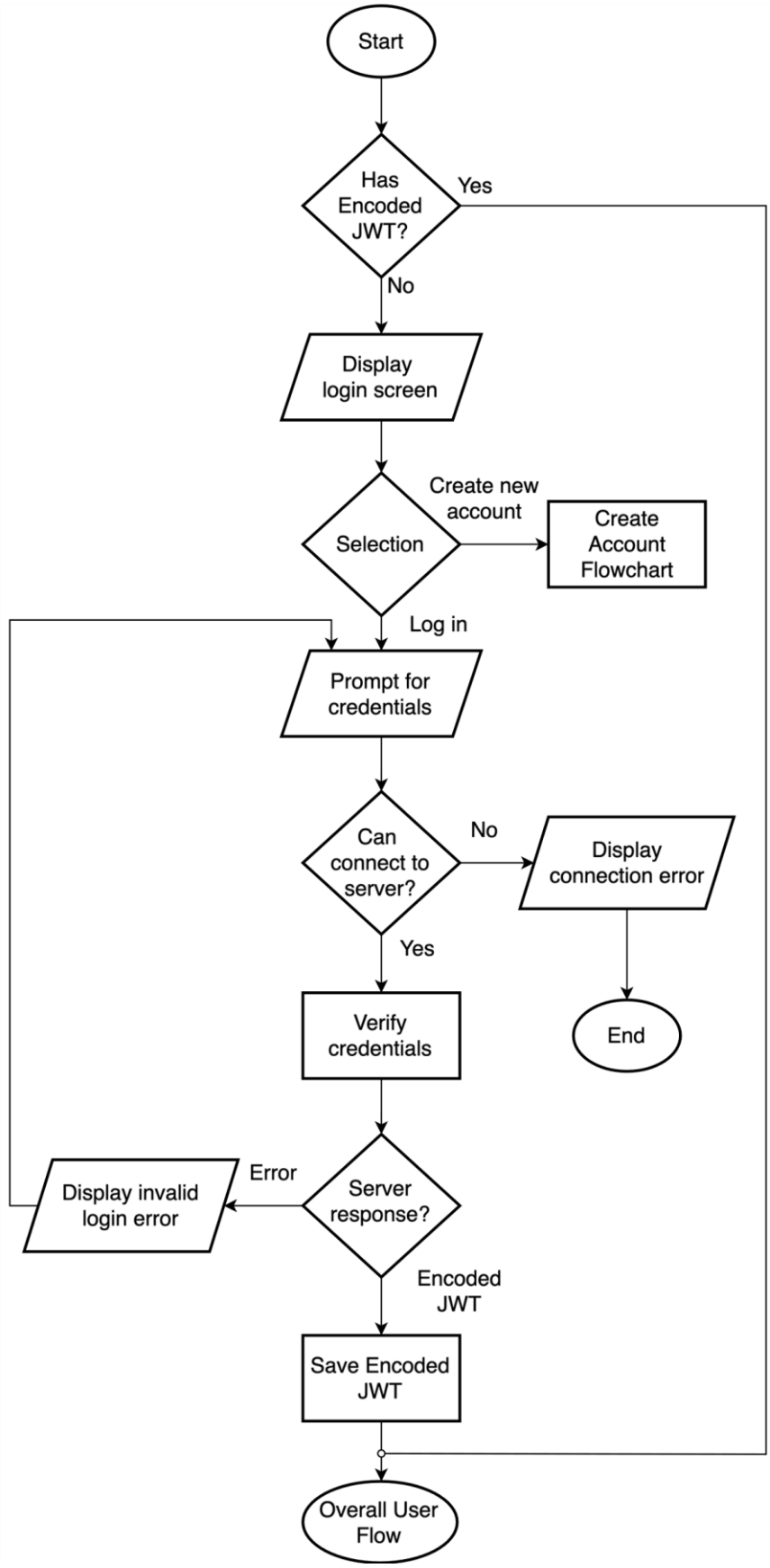
Name	#1.2.1e MainActivity View
Purpose	Serves as the primary activity of the **App**, managing the navigation between different fragments for the **User**.
Description	Serves as the central controller of the **App**'s main interface. It will initialize and manage a BottomNavigationView for navigating between the different sections of the **App**. MainActivity will load the appropriate fragment when a navigation item is selected.
Requirements	1
Elements	**bottomNavigation:** An instance of BottomNavigationView used to navigate between different sections of the **App**. (https://developer.android.com/reference/com/google/android/material/bottomnavigation/BottomNavigationView)
	sharedPreferences: A storage object used to save key-value pairs persistently, such as user login status or preferences. (https://developer.android.com/reference/android/content/SharedPreferences)
	loggedIn: A boolean variable indicating whether the **User** is currently logged in, is used to determine the **App**'s initial screen.
	onCreate: The main lifecycle method where MainActivity sets up its initial state, including layout setup, navigation initialization, and checking login status. Derived from AppCompatActivity. (https://developer.android.com/reference/androidx/appcompat/app/AppCompatActivity)
	loadFragment: A helper method used to load and replace fragments within the activity's view, allowing the **User** to switch between different screens.
	setUpNavigation: A method that initializes the bottomNavigation object, setting up listeners to load appropriate fragments when navigation items are selected.
	isUserLoggedIn: A method that determines if the **User's** login session is active.
Referenced By	**1 Overall Application**
Viewpoint	Class Diagram

View 1.2.1e.1: Main Activity Pseudocode View

```
1.
onCreate(savedState)
     super.OnCreate(savedState)
     IF isUserLoggedIn() returns false
          loginFragment ← Login(requestor)
          loadFragment(loginFragment)
     IF savedState is null
          homeFragment ← Home(requestor)
          loadFragment(HomeFragment)
     setUpNavigation()
END

2.
loadFragment(fragment)
     fragmentManager.beginTransaction()
     fragmentManager.replace(currentFragment, fragment)
     RETURN true
END

3.
setUpNavigation()
     localPlantsButton.onClick ← loadFragment(localPlantsFragment)
     allPlantsButton.onClick ← loadFragment(allPlantsFragment)
     settingsButton.onClick ← loadFragment(settingsFragment)
     homeButton.onClick ← loadFragment(homeFragment)
END
```

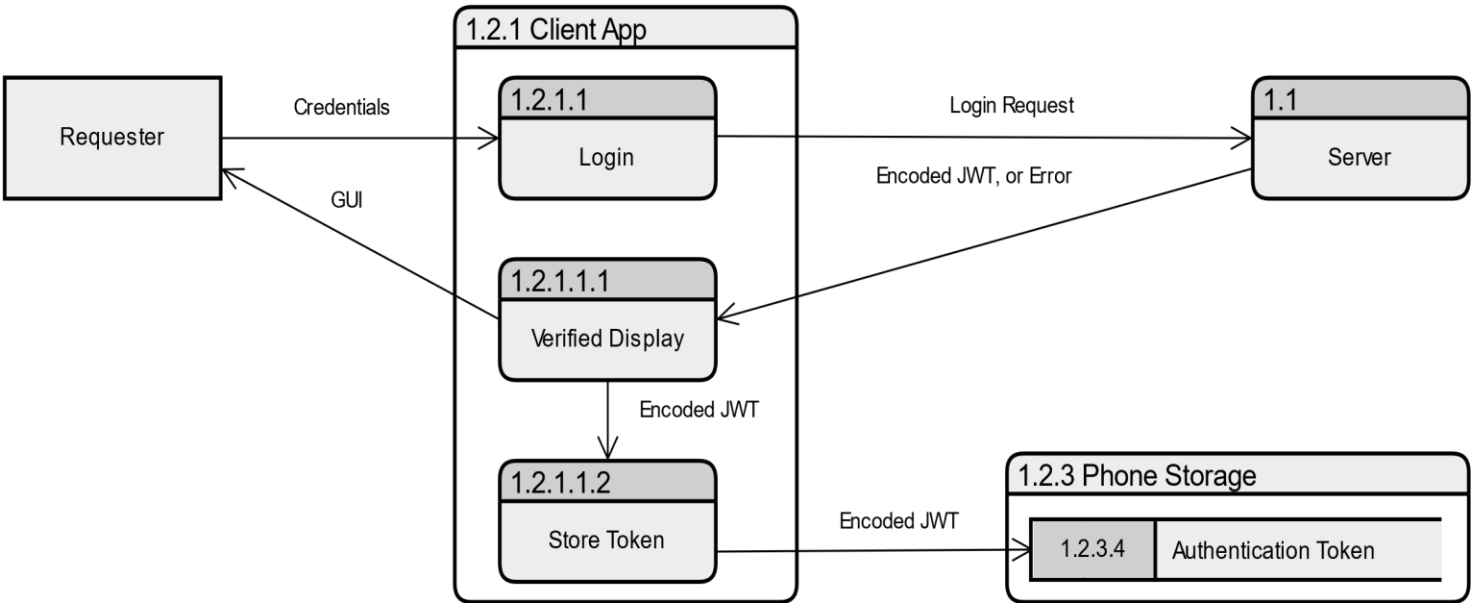
Name	1.2.1e.1 MainActivity Pseudocode View
Purpose	To outline the key methods for initializing and managing navigation and fragment transactions within the MainActivity Class.
Description	Provides pseudocode for interactions with the navigation bar, loading fragments, and initializing the **App.**
Requirements	1
Elements	**1. onCreate:** Main lifecycle method. Called when an instance of the MainActivity object is created.
	2. loadFragment: A helper method used to load and replace fragments.
	3. setUpNavigation: Initializes the bottomNavigation object by setting up listeners to load appropriate fragments when navigation items are selected.
Referenced By	**1.2.1e MainActivity Class Diagram**
Viewpoint	Pseudocode

View 1.2.1.1a: Login Process View



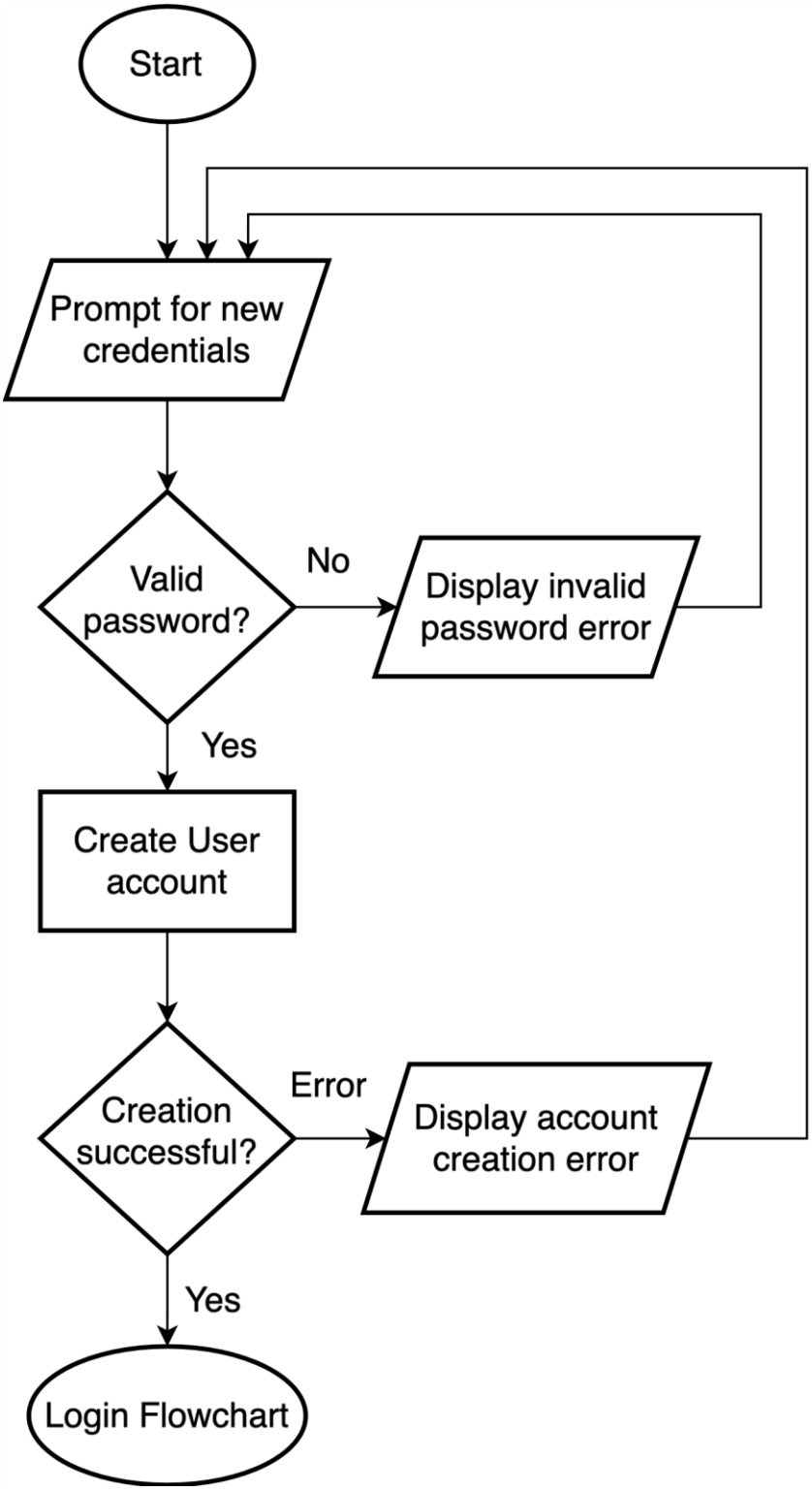
Name	#1.2.1.1a Login Process View
Purpose	Details the process of logging into the **App**.
Description	If the **User** has an existing Encoded JWT, then they can skip the Login process entirely. Otherwise, the **User** can choose to login using their **Credentials** or create a new account. The **App** cannot be accessed until the **User's Credentials** have been verified by the **Server**.
Requirements	19, 20, 21, 22
Elements	**1.2.1.3 Create Account Flowchart:** The process of creating a new **User** account.
	Verify Credentials: Verify that the given **Credentials** exist in the **Database**. Returns an error message if the username doesn't exist or the password doesn't match.
	1.2.1.1.2 Save Encoded JWT: Save the returned Encoded JWT to the device's local storage.
	1.1.2.2 Encoded JWT: Secure string that has encoded **User** authentication data.
	1.2.1b Overall User Flow: The flow of the **App** after the **User** has been **Authenticated**.
Referenced By	**1.2.1a Site Map View, 1.2.1b Overall User Flow Diagram View, 1.2.1.1b User Login Process Data Flow Diagram, 1.2.1.1c Authentication Token from Login View, 1.2.1.3 Create Account View**
Viewpoint	Flowchart

View 1.2.1.1b: User Login Process Data Flow Diagram



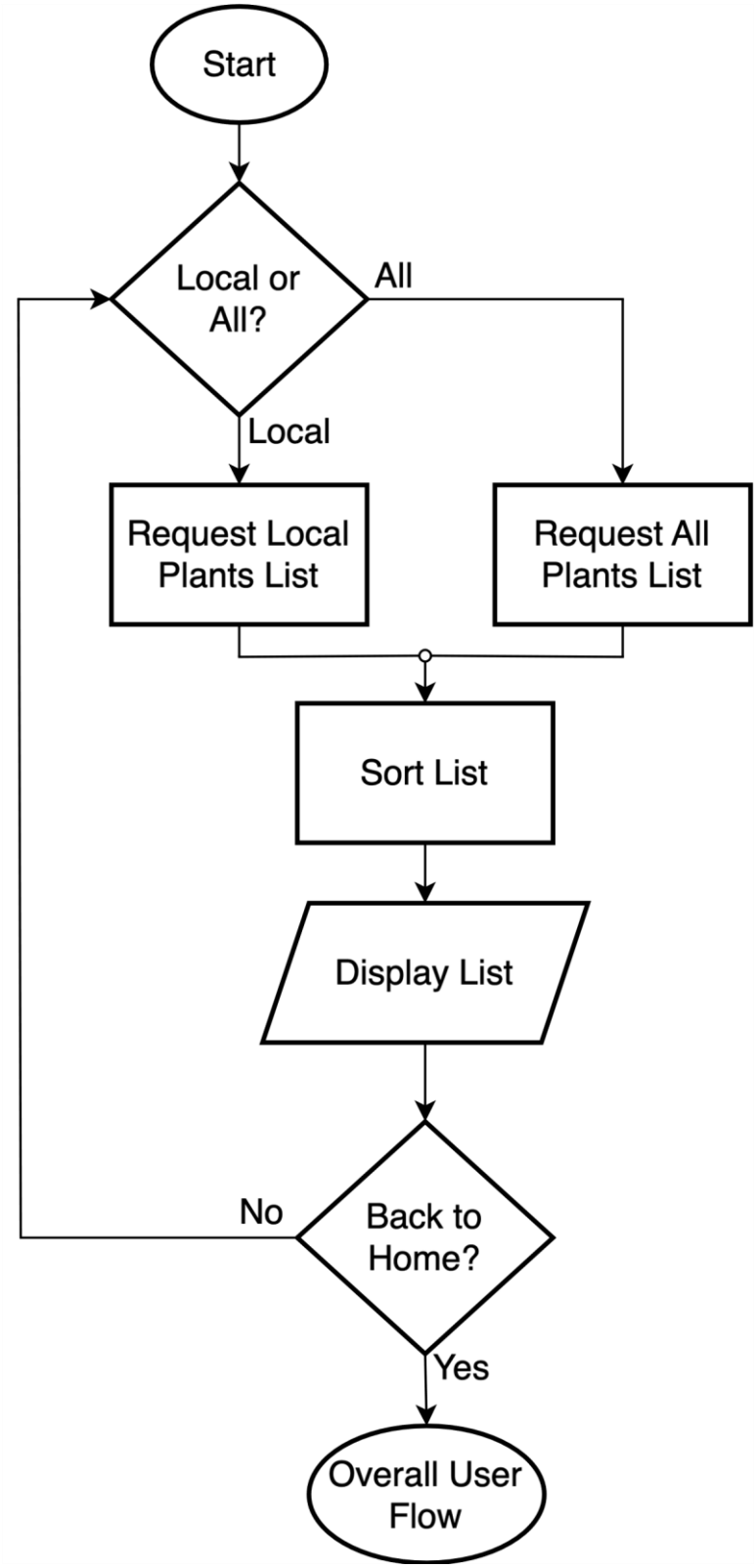
Name	#1.2.1.1b Authentication Token from Login
Purpose	This shows the process of storing a JSON Web Token (JWT) when a **User** logs into the **App.**
Description	The **Requestor** will provide the **App** with their **Credentials** to attempt to log into the **App**. The **App** will send a login request to the **Server**. Once the **Requestor** has been **Authenticated**, the **Server** will create a JWT and send it back to the **App**. The **App** will then store the JWT in the phone storage system.
Requirements	19, 21
Elements	**1.2.1.1 Login:** The **App** sends a login request, with the **User Credentials**, to the **Server** to be **Authenticated**.
	Verified Display: Displays to the **Requestor** if they have been verified or not.
	Store Token: The **App** receives an authentication JSON from the **Server** and proceeds to store it in the phone storage system.
	1.2.3.4 Authentication Token: Where the **Encoded JWT** is stored within the device's storage system.
	Credentials: The **Requestor**'s given login username and login password.
	Encoded JWT: An encoded string used for authentication.
	Login Request: The **Requestor's** username and password as strings, encrypted and then packaged and sent as an http request to the server.
Referenced By	**Login Flowchart**
Viewpoint	Data Flow Diagram

View 1.2.1.3: Create Account View



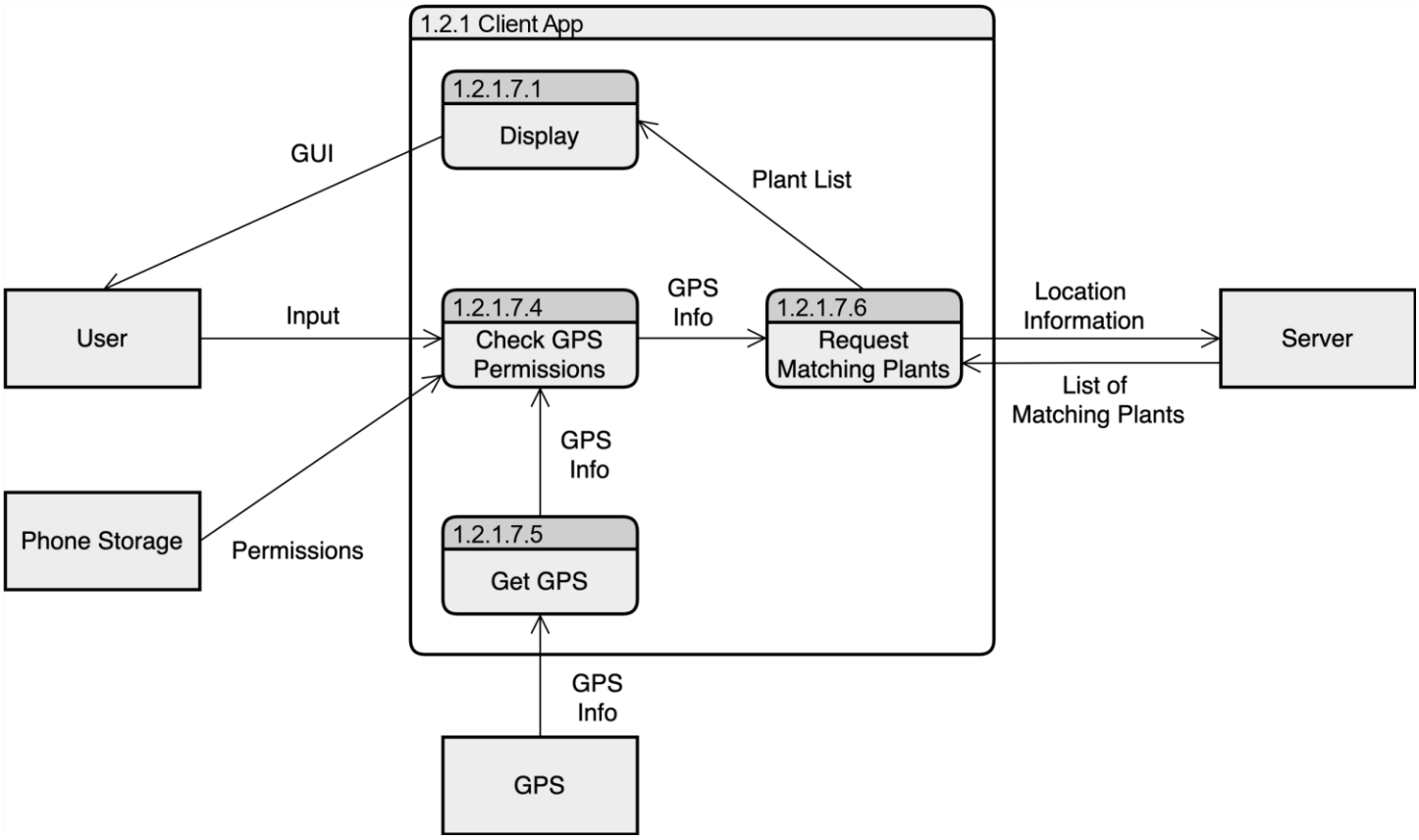
Name	#1.2.1.3 Create Account View
Purpose	To show the process of making a new **User** account.
Description	The **User** will navigate to the registration page from the login page their first time opening the **App**. The **User** is prompted for their **Credentials**. The **Credentials** entered by the **User** are checked, and if valid, a new **User** is created in the **Database**.
Requirements	20
Elements	**Valid password:** Checks if the password is complex enough.
	Create User Account: A new **User** is created in the **Database**. Returns an error if there is an issue creating the account.
	1.2.1.1a Login Flowchart: The flow of the **App** before the **User** has been **Authenticated**.
Referenced By	**1.2.1a Site Map View, 1.2.1.1a Login Process View**
Viewpoint	Flowchart

View 1.2.1.7: Find Plants Flowchart View


Start
Local or All?
All
Local
Request Local Plants List
Request All Plants List
Sort List
Display List
Back to Home?
No
Yes
Overall User Flow

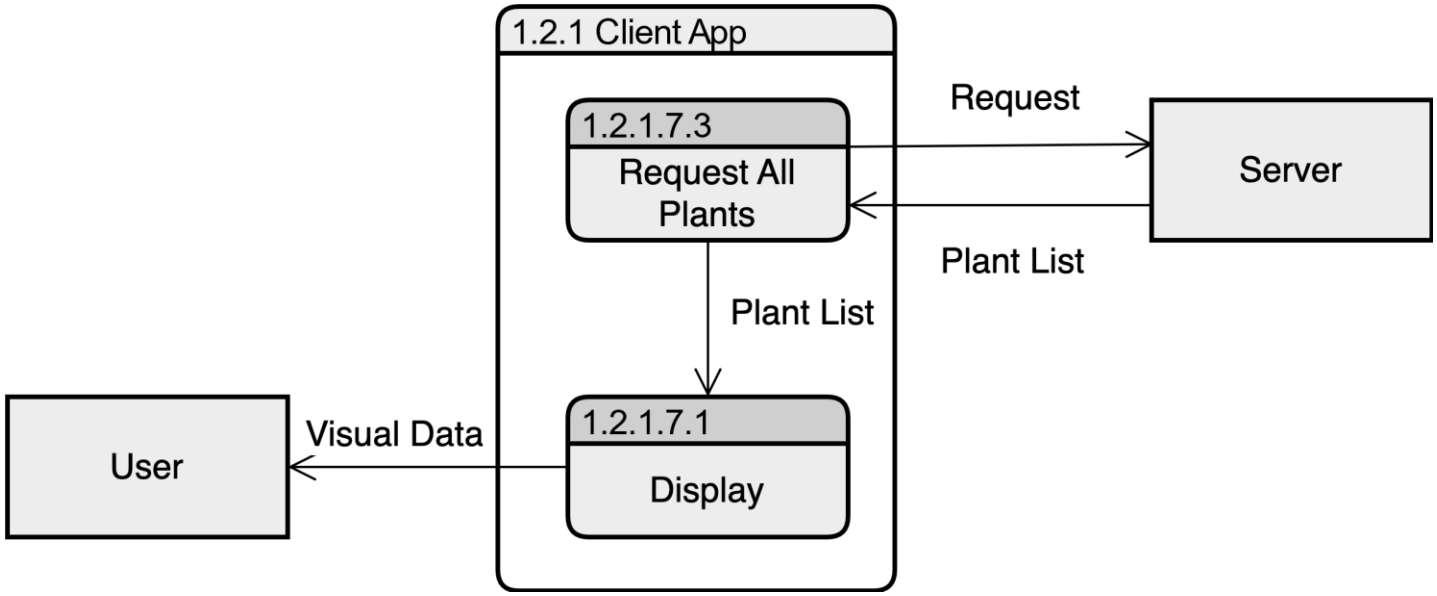
Name	#1.2.1.7 Find Plants Flowchart View
Purpose	To illustrate the process of requesting, processing, and displaying the **Local Plants** or **All Plants** list.
Description	The **User** can view **Local Plants** by enabling **Location** access or the **User** can view **All Plants** currently stored in the **Database**. The **User** can sort the **Plants** by name or **Use Type.**
Requirements	2, 3, 4, 5, 10
Elements	**Local or All:** A decision point in the **App** in which the **User** can choose whether to request the **All Plants** or the **Local Plants** list.
	1.2.1.7.2 Request Local Plants List: The **App** sends the **User**'s current **Location** to the **Server** as part of the request for retrieving the **Local Plants** list.
	1.2.1.7.3 Request All Plants List: The **App** initiates a request to the **Server** to retrieve the **All Plants List.** This request does not require **Location**.
	Sort List: Allows the **User** to sort the displayed **Plants** according to name or **Use Type.**
	1.2.1.7.1 Display List: Process of displaying a given list of **Plants** to the **User**.
	Back to Home: A decision point in the **App** in which the **User** can choose to go back to the Home page or stay on this page.
	1.2.1b Overall User Flow: The **User** navigates to the Home page of the **App**.
Referenced By	**1.2.1a Site Map View, 1.2.1b Overall User Flow Diagram**
Viewpoint	Flowchart

View 1.2.1.7.2: Request Local Plants View



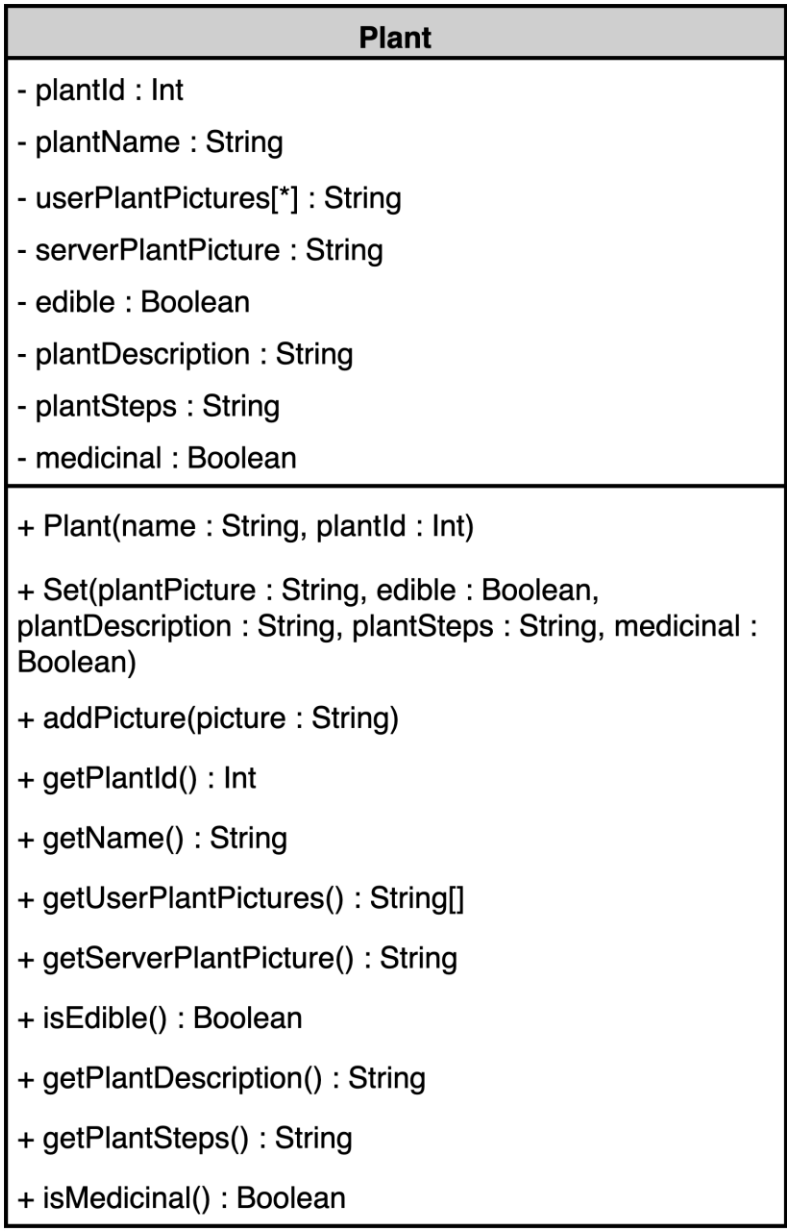
Name	#1.2.1.7.2 Get Local Plants View
Purpose	Retrieves a list of **Local Plants** through requests.
Description	When a request for **Local Plants** is made, it starts by requesting GPS location. To request GPS location, the **App** requires the **User's** authorization. Once approved, the **App** makes the requests for the GPS information. Upon receiving the GPS info the **App** will request a list of **Plants** from the **Server** that match the location of the GPS. Upon receiving said list it will be returned to the **User** to view.
Requirements	3, 4, 24
Elements	**1.2.1.7.4 Check GPS Permissions:** Check GPS will verify with the **User** that the **App** is allowed to access GPS services. As long as the **User** agrees, Check GPS will then send for Acquire GPS. If **User** does not agree then Check GPS returns to Request Local Plants with no information.
	1.2.1.7.5 Get GPS: Acquire GPS is where the **App** communicates with Phone Services to access the sub-system GPS. Requesting the GPS information.
	1.2.1.7.6 Request Matching Plants: Request Matching Plants is the point where the **App** communicates with the **Server** to retrieve a list of plants that match the GPS information given.
	Display: Display is the point where the plant list given is displayed. If no information is given by Request Local Plants because **User** did not allow Check GPS to access GPS services. Then Display will show a blank list.
	1.2.3 Phone Storage: Phone Storage View
	1.2.2 Phone Services:GPS: Get Location
	1.1 Server: Server DFD
Referenced By	**1.2.1.7 Find Plants Flowchart**
Viewpoint	Data Flow Diagram (DFD)

View 1.2.1.7.3: Request All Plants View



Name	#1.2.1.7.3 Get All Plants View
Purpose	Shows the process of retrieving the **All Plants List** from the **Server**.
Description	A request for **All Plants List** is made to the **Server** and then displayed to the **User**.
Requirements	10
Elements	**Request All Plants:** Request **All Plants List** from the **Server**. Converts the retrieved JSON data into a list of **Plants**.
	Display: Sorts and displays the **All Plants List** to the **User**.
	Request: An HTTP request sent to the **Server** to request a list of all the **Plants**.
Referenced By	**1.2.1.7 Find Lists Flowchart**
Viewpoint	Data Flow Diagram (DFD)

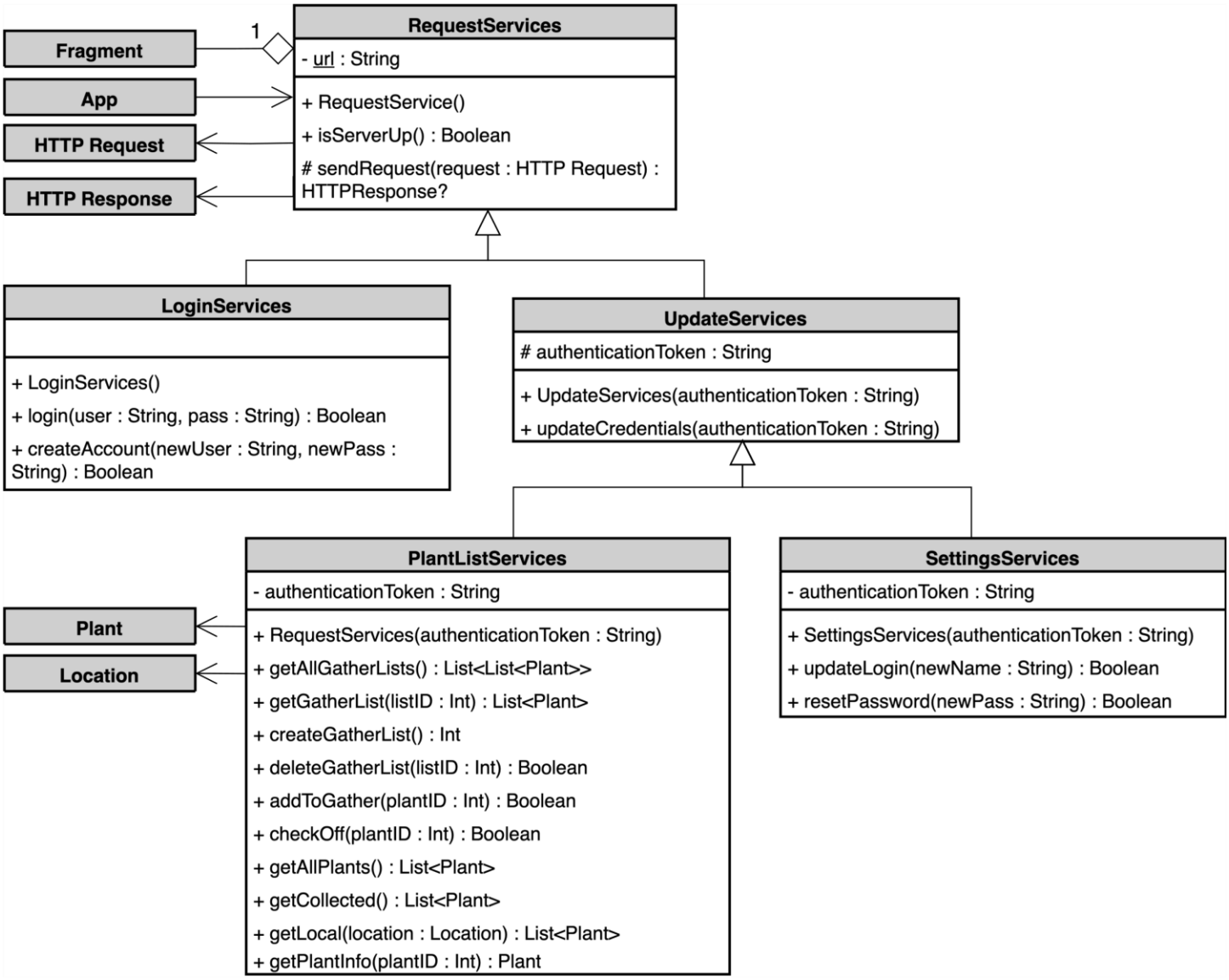
View 1.2.1.8: Plant Class Diagram View



Name	#1.2.1.8 Plant Class Diagram View
Purpose	Shows how a **Plant** object is stored in the **App**.

Description	The data received from the **Server** when a request for a **Plant** is made.
Requirements	7, 11
Elements	**Primary Constructor:** Creates a **Plant** object from just the **Plant** name and id received from the **Database**. Sets all other attributes to null.
	Set: Sets all attributes to the data received from the **Database** after a request is made.
	1.2.1.8.1 Add Picture: Adds a **Photo** to the list of userPlantPictures.
	Plant ID: Unique ID for the given **Plant**.
	Plant Name: The name of the **Plant** retrieved from the **Database.**
	User Plant Pictures: A list of strings representing the file path stored on the device for each **Plant** photo.
	Server Plant Picture: A string representing the varbinary of a **Photo** of the given **Plant** received from the **Database**.
	Edible: Indicates if the given **Plant** is edible.
	Plant Description: A short description of the given **Plant**.
	Plant Steps: The sequence of **Steps** to identify the **Plant**.
	Medicinal: Indicates if the **Plant** can be used for medicine.
Referenced By	**1.2.1a Client Application Site Map View, 1.2.1.16.5 PlantInfo Class Diagram View**
Viewpoint	Class Diagram

View 1.2.1.9: Request Services Class Diagram View


Fragment
App
HTTP Request
HTTP Response
1
RequestServices
- url : String
+ RequestService()
+ isServerUp() : Boolean
# sendRequest(request : HTTP Request) : HTTPResponse?
LoginServices
+ LoginServices()
+ login(user : String, pass : String) : Boolean
+ createAccount(newUser : String, newPass : String) : Boolean
UpdateServices
authenticationToken : String
+ UpdateServices(authenticationToken : String)
+ updateCredentials(authenticationToken : String)
Plant
Location
PlantListServices
- authenticationToken : String
+ RequestServices(authenticationToken : String)
+ getAllGatherLists() : List<List<Plant>>
+ getGatherList(listID : Int) : List<Plant>
+ createGatherList() : Int
+ deleteGatherList(listID : Int) : Boolean
+ addToGather(plantID : Int) : Boolean
+ checkOff(plantID : Int) : Boolean
+ getAllPlants() : List<Plant>
+ getCollected() : List<Plant>
+ getLocal(location : Location) : List<Plant>
+ getPlantInfo(plantID : Int) : Plant
SettingsServices
- authenticationToken : String
+ SettingsServices(authenticationToken : String)
+ updateLogin(newName : String) : Boolean
+ resetPassword(newPass : String) : Boolean

Name	#1.2.1.9 Request Services Class Diagram View
Purpose	Shows the RequestServices class and derived children classes which contain attributes and methods associated with **Server** communication.
Description	This class will be created and given to Fragments in order to handle communication with the **Server**. Requests and responses will be sent as HTTP Messages, with only relevant data from **Server** responses being returned when methods are called.
Requirements	1, 14, 15, 16, 17, 18, 19, 20, 21, 22
Elements	**HTTP Messages:** HTTP Requests and HTTP Responses which are used to communicate between the **App** and the **Server**. (https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages)
	RequestServices: The base class which handles communication with the **Server**. All requests will be sent the same way.
	url: The address of the **Server**, where all HTTP Requests will be sent.
	Request::isServerUp: Returns true if the **Server** can be reached.
	RequestServices::sendRequest: Sends the given HTTP Request to the **Server**, returning the HTTP Response.
	LoginServices: Creates and handles requests from the Login page before the **User** has been **Authenticated**.
	LoginServices::login: Sends a request to **Authenticate** a **User** using the given **Credentials**. Returns true if the **User** was successfully **Authenticated**. The username and JWT will be assigned from the **Server's** response.
	LoginServices::createAccount: Sends a request to create a new account. Returns true if the account was created successfully.
	UpdateServices: Creates and handles requests after the **User** has been **Authenticated**. Includes the User's authenticationToken when creating HTTP Requests.
	1.2.3.4 authenticationToken: The string used by the **Server** to identify the **User** in order to retrieve their data.
	UpdateServices::updateCredentials: Allows the authenticationToken to be updated.
	SettingsServices: Creates and handles requests relating to updating the **User's** account.

	SettingsServices::updateLogin: Updates the username and authentication token. These variables will be used in HTTP Requests to **Authenticate** the **User**.
	SettingsServices::resetPassword: Sends a request to update the given **User's** password. Returns true if the password has been updated.
	PlantListServices: Creates and handles requests relating to retrieving **Plant** lists created by the **User**.
	PlantListServices::getAllGatherList: Sends a request to retrieve all of the **User's Gather Lists**. Converts the response into a list of **Plants** lists.
	PlantListServices::getGatherList: Sends a request to retrieve one of the **User's Gather List**. Converts the response into a list of **Plants**.
	PlantListServices::createGatherList: Sends a request to create a new **Gather List**. Returns the new list's id, or -1 if there was an error creating a new list.
	PlantListServices::deleteGatherList: Sends a request to delete a **User's Gather List.** Returns true if the list is deleted successfully.
	PlantListServices::addToGather: Sends a request to add the given **Plant** to the selected **User's Gather List**. Returns if the **Gather List** was successfully updated.
	PlantListServices::checkOff: Sends a request to move the given **Plant** from one of the **User's Gather Lists** to the **User's Collected List**. Returns if the lists were successfully updated.
	PlantListServices::getAllPlants: Sends a request to retrieve the full list of **Plants**. Converts the response into a list of **Plants**.
	PlantListServices::getCollectedList: Sends a request to retrieve the **User's Collected List**. Converts the response into a list of **Plants**.
	PlantListServices::getLocalList: Sends a request to retrieve a list of **Plants** around the **User's** location. Converts the response into a list of **Plants**. The data is passed as a Location, a class containing the latitude, longitude, and other data. (https://developer.android.com/reference/android/location/Location)
	PlantListServices::getPlantInfo: Sends a request to retrieve info about the given **Plant**. Returns the **Plant** data as a class.
Referenced By	**1.2.1e Main Activity View**
Viewpoint	Class Diagram

View 1.2.1.9.1: Request Services Pseudocode View

```
.1
RequestServices::sendRequest(request)
     TRY
          client ← OkHttpClient
          response ← client.newCall(request).execute()
          RETURN response
     CATCH network error
     CATCH IO exception
     CATCH exceptions

     RETURN NULL

.2
RequestServices::isServerUp()
     request ← Request.Builder()
     request.method ← “GET”
     request.url ← url
     request.build()

     RETURN sendRequest(request) not NULL

.3
LoginServices::login(username, password)
     request ← Request.Builder()
     request.method ← “GET”
     request.url ← url += “/login”
     request.body ← { “username” : username, “password : password }

     request.build()

     response ← sendRequest(request)
```

```
     IF response not NULL AND response.body() not NULL
          returnJSON ← response.body()[“data”] as a dictionary
          IF returnJSON[“success”]
               updateLogin(returnJSON[“userID”], returnJSON[“authToken”])
               RETURN TRUE
          RETURN FALSE
     RETURN FALSE
```

.4

```
LoginServices::createAccount(newUser, newPass)
     request ← Request.Builder()
     request.method ← “POST”
     request.url ← url += “/registration”
     request.body ← {“username” : newUser, “password” : newPass }
     request.build()

     response ← sendRequest(request)

     RETURN response not NULL AND response.body()[“data”][“success”]
```

.5

```
SettingsServices::updateLogin(newName)
     request ← Request.Builder()
     request.method ← “POST”
     request.url ← url += “/user/” += newName
     request.headers ←     “authorization” : authToken
     request.build()

     response ← sendRequest(request)

     RETURN response not NULL AND response.body()[“data”][“success”]
```

.6

```
SettingsServices::resetPassword(newPass)
```

```
     request ← Request.Builder()
     request.method ← “POST”
     request.url ← url += “/user”
     request.headers ←     “authorization” : authToken
     request.body ← newPass
     request.build()

     response ← sendRequest(request)

     RETURN response not NULL AND response.body()[“data”][“success”]
```

.7

```
PlantListServices::getAllGatherList()
     request ←  Request.Builder()
     request.method ← “GET”
     request.url ← url += “/gather”
     request.headers ←     “authorization” : authToken
     request.build()

     response ← sendRequest(request)

     IF response not NULL AND response.body() not NULL
          returnJSON ← response.body()[“data”] as a dictionary
          allGatherList ← returnJSON[“plantList”] as a collection of Gather Lists
(Map<String, List<Plant>>)
          RETURN allGatherList
     RETURN NULL
```

.8

```
PlantListServices::getGatherList(listID)
     request ←  Request.Builder()
     request.method ← “GET”
     request.url ← url += “/gather/” += listID
     request.headers ←     “authorization” :  authToken
```

```
     request.build()

     response ← sendRequest(request)

     IF response not NULL AND response.body() not NULL
          returnJSON ← response.body()[“data”] as a dictionary
          gatherList ← returnJSON[“plantList”] as a List<Plant>
          RETURN gatherList
     RETURN NULL
```

.9

```
PlantListServices::createGatherList()
     request ← Request.Builder()
     request.method ← “POST”
     request.url ← url += “/gather/” += newListID
     request.headers ←     “authorization” : authToken
     request.build()

     response ← sendRequest(request)

     IF response not NULL AND request.body() not NULL
           returnJSON ← response.body()[“data”] as a dictionary
           RETURN returnJSON[“listID”]
     RETURN -1
```

.10

```
PlantListServices::deleteGatherList(listID)
     request ←  Request.Builder()
     request.method ← “DELETE”
     request.url ← url += “/gather/” + listID
     request.headers ←     “authorization” : authToken
     request.build()

     response ← sendRequest(request)
```

```
     RETURN response not NULL AND response.body()[“data”][“success”]

.11
PlantListServices::addToGather(listID, plantID)
     request ← Request.Builder()
     request.method ← “POST”
     request.url ← url += “/gather/” += listId += “/” += plantID
     request.headers ←     “authorization” : authToken
     request.build()

     response ← sendRequest(request)

     RETURN response not NULL AND response.body()[“data”][“success”]

.12
PlantListServices::checkOff(listID, PlantID)
     request ← Request.Builder()
     request.method ← “DELETE”
     request.url ← url += “/gather/” += listID += “/” += plantID
     request.headers ←     “authorization” : authToken
     request.build()

     response ← sendRequest(request)
     IF response not NULL AND response.body() not NULL
           RETURN response.body()[“data”][“success”]
     RETURN FALSE


.13
PlantListServices::getAllPlants()
     request ← Request.Builder()
     request.method ← “GET”
     request.url ← url += “/plant”
```

```
    request.header ← “authorization” : authToken
    request.build()

    response ← sendRequest(request)

    IF response not NULL AND response.body() not NULL
        returnJSON ← response.body()[“data”] as a dictionary
        allPlantList ← returnJSON[“plantList”] as a List<Plant>
        RETURN allPlantList
    RETURN NULL
```

.14

```
PlantListServices::getCollectedList()
    request ←  Request.Builder()
    request.method ← “GET”
    request.url ← url += “/collected”
    request.headers ← “authorization” : authToken
    request.build()

    response ← sendRequest(request)

    IF response not NULL AND response.body() not NULL
        returnJSON ← response.body()[“data”] as a dictionary
        collectedPlants ← returnJSON[“plantList”] as a List<Plant>
        RETURN collectedList
    RETURN NULL
```

.15

```
PlantListServices::getLocalList(location)
    request ←  Request.Builder()
    request.method ← “GET”
    request.url ← url += “/localPlants/” + location
    request.headers ← “authorization” : authToken
    request.build()
```

```
     response ← sendRequest(request)

     IF response not NULL AND response.body() not NULL
          returnJSON ← response.body()[“data”] as a dictionary
          localPlants ← returnJSON[“plantList”] as a List<Plant>
          RETURN localPlants
     RETURN NULL
```

.16

PlantListServices::getPlantInfo(plantID)

```
     request ←  Request.Builder()
     request.method ← “GET”
     request.url ← url += “/plant/” += plantID
     request.header ←      “authorization” : authToken
     request.build()

     response ← sendRequest(request)

     IF response not NULL AND response.body() not NULL
          returnJSON ← response.body()[“data”] as a dictionary
          plant ← returnJSON[“plant”] as a Plant
          RETURN plant
     RETURN NULL
```

Name	#1.2.1.9.1 Request Services Pseudocode View
Purpose	Shows the pseudocode of the Request Services class.
Description	Details the pseudocode for several methods within the Request Services class. This class handles interactions between the client and the **Server**.
Requirements	1, 14, 15, 16, 17, 18, 19, 20, 21, 22
Elements	**.1 RequestServices::sendRequest:** Sends the given HTTP Request to the **Server**, returning the HTTP Response.
	.2 Request::isServerUp: Returns true if the **Server** can be reached.
	.3 LoginServices::login: Sends a request to **Authenticate** a **User** using the given **Credentials**. Returns true if the **User** was successfully **Authenticated**. The username and JWT will be assigned from the **Server's** response.
	.4 LoginServices::createAccount: Sends a request to create a new account. Returns true if the account was created successfully.
	.5 SettingsServices::updateLogin: Updates the username and authentication token. These variables will be used in HTTP Requests to **Authenticate** the **User**.
	.6 SettingsServices::resetPassword: Sends a request to update the given **User's** password. Returns true if the password has been updated.
	.7 PlantListServices::getAllGatherList: Sends a request to retrieve all of the **User's Gather Lists**. Converts the response into a list of **Plants** lists.
	.8 PlantListServices::getGatherList: Sends a request to retrieve one of the **User's Gather List**. Converts the response into a list of **Plants**.
	.9 PlantListServices::createGatherList: Sends a request to create a new **Gather List**. Returns the new list's id, or -1 if there was an error creating a new list.
	.10 PlantListServices::deleteGatherList: Sends a request to delete a **User's Gather List.** Returns true if the list is deleted successfully.
	.11 PlantListServices::addToGather: Sends a request to add the given **Plant** to the selected **User's Gather List**. Returns if the **Gather List** was successfully updated.
	.12 PlantListServices::checkOff: Sends a request to move the given **Plant** from one of the **User's Gather Lists** to the **User's Collected List**. Returns if the lists were successfully updated.
	.13 PlantListServices::getAllPlants: Sends a request to retrieve the full list of **Plants**. Converts the response into a list of **Plants**.

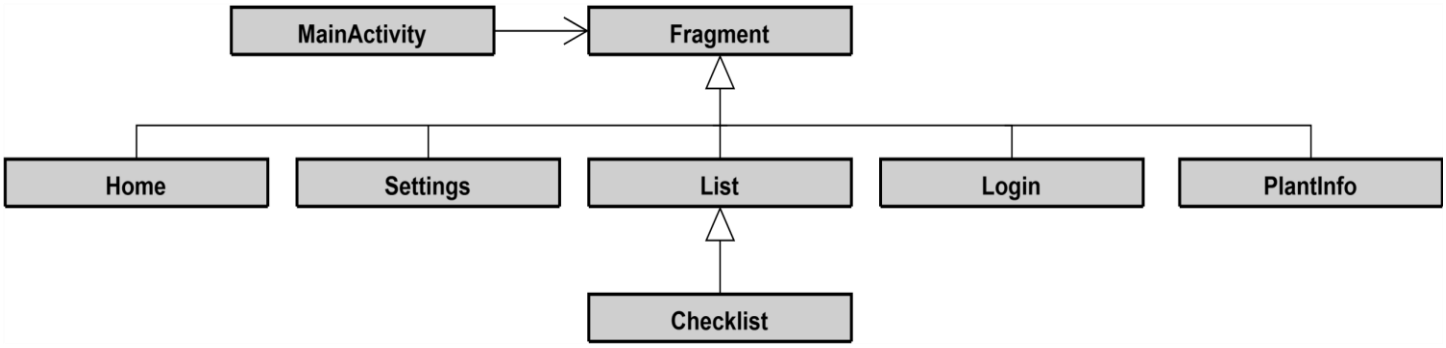
	.14 PlantListServices::getCollectedList: Sends a request to retrieve the **User's Collected List**. Converts the response into a list of **Plants**.
	.15 PlantListServices::getLocalList: Sends a request to retrieve a list of **Plants** around the **User's** location. Converts the response into a list of **Plants**. The data is passed as a Location, a class containing the latitude, longitude, and other data. (https://developer.android.com/reference/android/location/Location)
	.16 PlantListServices::getPlantInfo: Sends a request to retrieve info about the given **Plant**. Returns the **Plant** data as a class.
Referenced By	**1.2.1.9 Request Services Class Diagram**
Viewpoint	Pseudocode

View 1.2.1.13 Store Photo ID Json Pseudocode

```
takePhoto(plantID)
// Take Photo Code
     photoID ← contentValues.getPhotoPath()
     READ photosJSON
          photosJSON[plantID].append(photoID)
     WRITE photosJSON
```

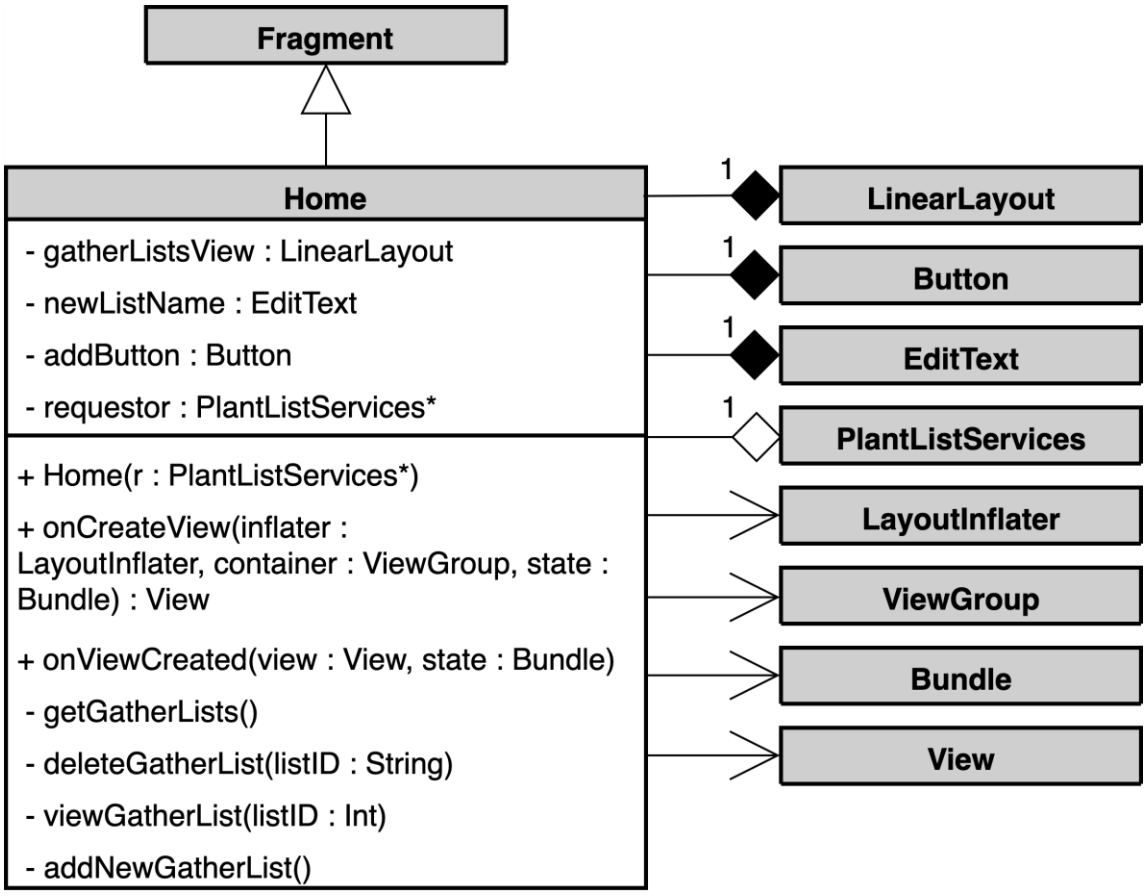
Name	#1.2.1.13 Store Photo ID JSON Pseudocode
Purpose	To describe the way a **User Photo** is given a unique identifier and how that identifier is added to storage.
Description	This diagram describes a function call of taking a **Photo**, given the **plant ID** we are taking the **Photo** for. We then save the **Photo's** file path into a preformatted JSON file.
Requirements	12, 13
Elements	**plantID:** The unique identifier of the **Plant** that we are taking a picture of.
	Take Photo Code: Predefined code for taking a **Photo** in our **App** by android. (https://developer.android.com/codelabs/camerax-getting-started#4)
	photoID: The unique identifier for the taken picture, which is also the file path for that **Photo**.
	contentValues: Information generated about the **Photo** as the **Photo** is taken by the predefined **Take Photo Code**.
	photosJSON: the filename of the file where the entire list of **Plant Photos** is stored as a JSON object.
Referenced By	**1.2.1d User Takes Photo Data Flow Diagram**
Viewpoint	Pseudocode

View 1.2.1.16: Fragment Hierarchy Class Diagram View



Name	1.2.1.16 Fragment Hierarchy View
Purpose	Serves as an overview of the Fragment class and its derived classes, illustrating the inheritance structure and relationships between the base class and specialized child classes.
Description	The base Fragment class, provided by Kotlin, serves as the foundation for our **App**'s navigation. From this class, we derive five primary fragments, each handling a specific page in the **App**. Additionally, the ListFragment class acts as a base for four specialized fragments, each corresponding to a distinct type of collection the **User** can view.
Requirements	4, 6, 7, 10, 14, 15, 23
Elements	**1.2.1e MainActivity:** The main activity class that is in charge of hosting the fragments.
	Fragment: The base class provided by Kotlin, used to define reusable and modular components for user interface and logic within the **App**. (https://developer.android.com/guide/fragments)
	1.2.1.16.1 Home: A fragment that displays the main page of the **App**.
	1.2.1.16.2 Settings: A fragment that allows the **User** to view and modify **User Settings**.
	1.2.1.16.3 List: A base fragment that provides shared functionality and UI structure for displaying different types of collections.
	1.2.1.16.3 Checklist: A fragment that displays a checklist.
	1.2.1.16.4 Login: A fragment that handles the login and registration process.
	1.2.1.16.5 PlantInfo: A fragment that displays detailed information about a specific **Plant**.
Referenced By	**1.2.1e Main Activity Diagram**
Viewpoint	Class Diagram

View 1.2.1.16.1: Home Class Diagram View



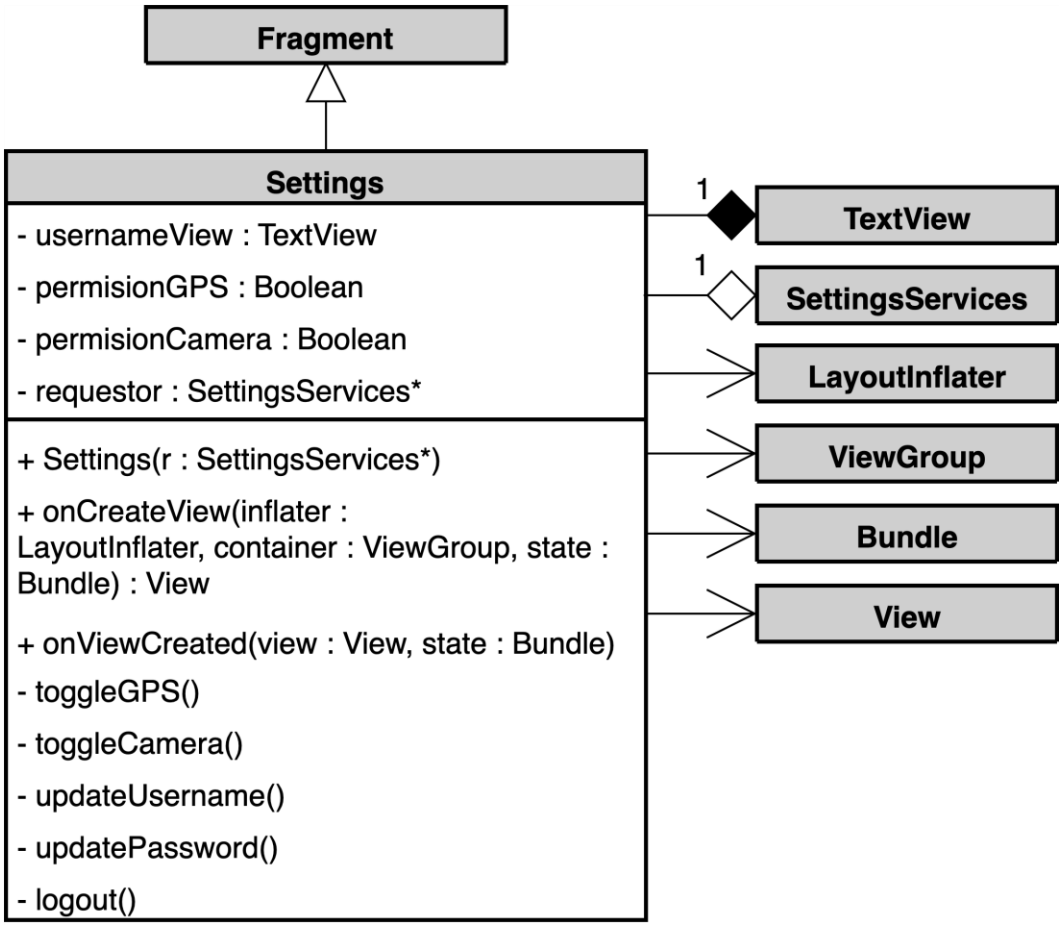
Name	1.2.1.16.1 Home Class Diagram View
Purpose	Serves as an overview of the Home class, which extends the Fragment class.
Description	Class that represents the Home screen for the **App** as well as their methods and attributes.
Requirements	4, 6, 7, 10, 13, 14, 15, 23
Elements	**gatherListsView:** A Linear Layout object that will be populated with the **User's Gather Lists** in getGatherLists. (https://developer.android.com/reference/android/widget/LinearLayout)
	newListName: An EditText that the **User** can enter the name for a new **Gather List**.
	addButton: The button the **User** taps to add a new Gather List using the name in the newListName EditText.
	requestor: A helper class that will handle communication with the **Server**.
	onCreateView: A lifecycle method in each class that is called to inflate the fragment's layout and the UI components are set up. (https://developer.android.com/reference/android/app/Fragment#onCreateView(android.view.LayoutInflater,%20android.view.ViewGroup,%20android.os.Bundle))
	onViewCreated: A lifecycle method in each class that is called after onCreateView and performs the actual logic for the **User** interacting with the UI elements. (https://developer.android.com/reference/android/app/Fragment#onViewCreated(android.view.View,%20android.os.Bundle)
	getGatherLists: A method that dynamically sets up the Linear Layout with all of the **User's Gather Lists.**
	deleteGatherList: A method that deletes the specified **Gather List.**
	viewGatherList: A method that will load a Checklist fragment for the specified **Gather List**.
	addNewGatherList: A method that will create a new **Gather List** for the **User**.
Referenced By	**1.2.1.16 Fragment Inheritance Map**
Viewpoint	Class Diagram

View 1.2.1.16.1.1: Home Fragment Pseudocode View

```
1.
getGatherList()
     allGatherLists ← requestor.getAllGatherLists()
     FOR each key in allGatherLists.keys
          gatherListMap[key] ← allGatherLists[key]
     PUT gatherListMap


2.
deleteGatherList(listID)
     requestor.deleteGatherList(listID)
     DELETE gatherListMap[listID]


3.
viewGatherList(listID)
      gatherList ← requestor.getGatherList(listID)

      MainActivity.loadFragment(CheckList(gatherList))


4.
addNewGatherList()
      listName ← newListName.text

      IF listName not in gatherListMap.keys
            requestor.createGatherList(listName)
            gatherListMap[listName] ← listName : new List<Plant>
```

Name	#1.2.1.16.1.1 Home Fragment Pseudocode View
Purpose	Shows the pseudocode of the Home fragment.
Description	Shows the pseudocode for methods within the Home fragment. This fragment displays buttons that allow the **User** to navigate to their **Gather Lists**, **Tick List**, **Local Plants List**, or the **All Plants List**.
Requirements	4, 6, 7, 10, 13, 14, 15, 23
Elements	**getGatherLists:** A method that dynamically sets up the Linear Layout with all of the **User's Gather Lists.**
	deleteGatherList: A method that deletes the specified **Gather List.**
	viewGatherList: A method that will load a Checklist fragment for the specified **Gather List**.
	addNewGatherList: A method that will create a new **Gather List** for the **User**.
	requestor.deleteGatherList: Sends a request to delete the specified **Gather List** on the **Server**.
	requestor.getAllGatherList: Sends a request to retrieve all of the current **User's Gather Lists** from the **Server**.
Referenced By	**1.2.1.16.1 Home Fragment Class Diagram**
Viewpoint	Pseudocode

View 1.2.1.16.2: Settings Class Diagram View



Name	1.2.1.16.2 Settings Class Diagram View
Purpose	Serves as an overview of the Settings class, which extends the Fragment base class.
Description	Class that represents the Settings screen for the **App** as well as its methods and attributes.
Requirements	4, 6, 7, 10, 13, 14, 15, 23
Elements	**onCreateView:** A lifecycle method in each class that is called to inflate the fragment's layout and the UI components are set up. (https://developer.android.com/reference/android/app/Fragment#onCreateView(android.view.LayoutInflater,%20android.view.ViewGroup,%20android.os.Bundle))
	onViewCreated: A lifecycle method in each class that is called after onCreateView and performs the actual logic for the **User** interacting with the UI elements. (https://developer.android.com/reference/android/app/Fragment#onViewCreated(android.view.View,%20android.os.Bundle)
	usernameView: An TextView object that holds the account username. (https://developer.android.com/reference/android/widget/TextView)
	permissionGPS: A boolean variable that indicates whether the **User** has permission to access the phone's GPS services.
	permissionCamera: A boolean variable that indicates whether the **User** has allowed permission to access the phone's camera services.
	requestor: A helper class that will handle communication with the **Server**.
	toggleGPS: A method that allows the **User** to toggle the **App** permissions to retrieve **GPS Location**.
	toggleCamera: A method that allows the **User** to toggle the **App** permissions to access the **Camera**.
	updateUsername: A method that allows the **User** to update their username.
	updatePassword: A method that allows the **User** to update their password.
	logout: A method that allows the **User** to log out of the **App**.
Referenced By	**1.2.1.16 Fragment Inheritance Map**
Viewpoint	Class Diagram

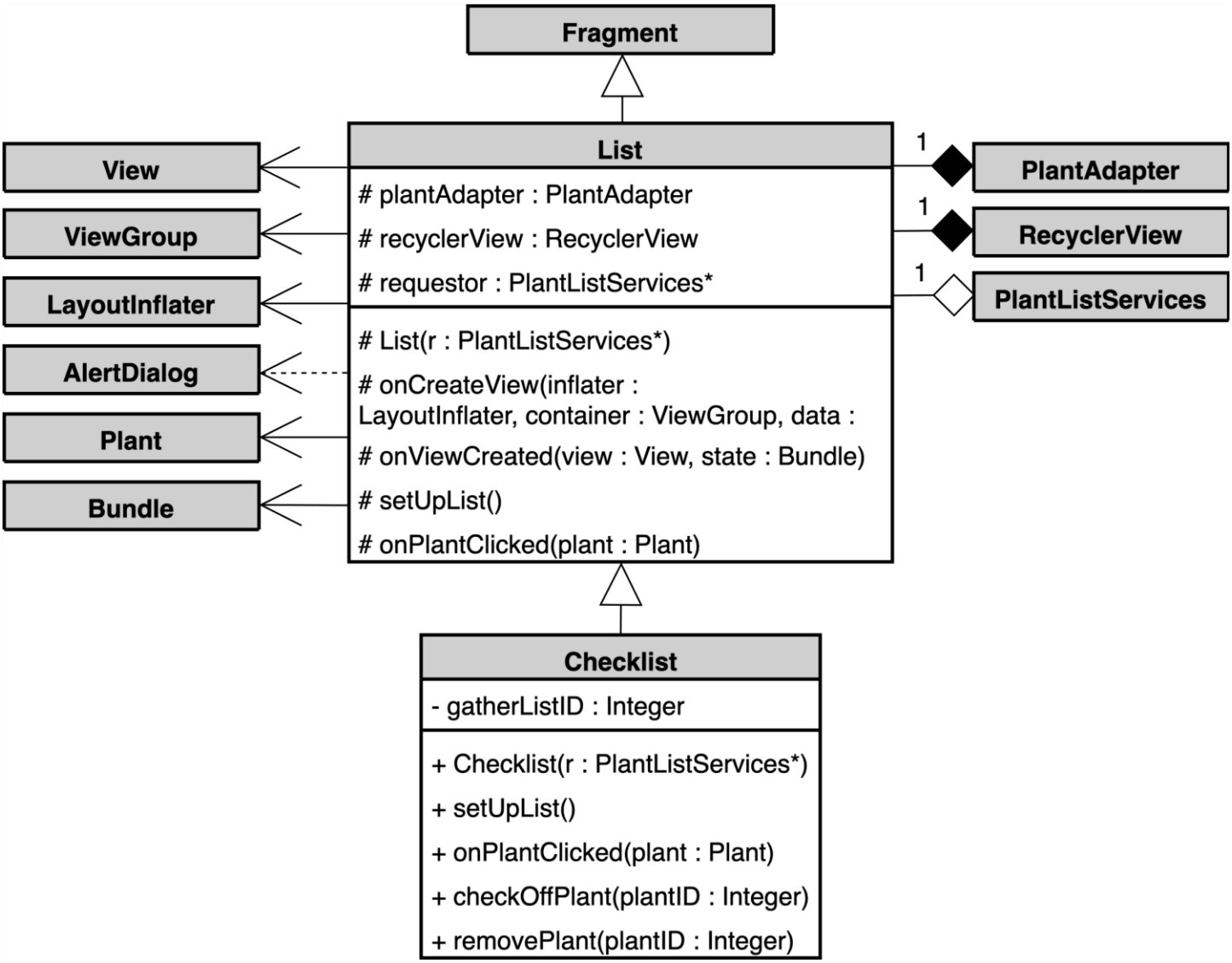
View 1.2.1.16.2.1: Settings Pseudocode View

```
1.
toggleGPS()
     pSPath ← “PhoneStorageSourceFilePath”
     permissions ← class inside file at psPath
     #tell server of the change so it’s saved
      permissionGPS ← permissions.checkSelfPermission(context, GPS)
     IF permissionGPS = false
          permissionGPS ← true
          permissions.requestPermission()
     ELSE
          permissionGPS ← false

2.
toggleCamera()
     pSPath ← “PhoneStorageSourceFilePath”
     Permissions ← class inside file at pSPath
     #tell server of the change so it’s saved
      permissionCamera ← permissions.checkSelfPermission(context, Camera)
     IF permissionCamera = false
          permissionCamera ← true
          permissions.requestPermission()
     ELSE
          permissionCamera ← false


3.
updateUsername()
     GET newUserName
     requestor.updateLogin(newUserName)

4.
updatePassword()
     GET newPassword
```

```
requestor.resetPassword(newPassword)
```

Name	#1.2.1.16.2.1 Settings Pseudocode View
Purpose	Shows the pseudocode of the Settings fragment.
Description	Shows the pseudocode for methods within the Settings fragment. Allows the **User** to update app permissions, update their username or password, or log out.
Requirements	4, 6, 7, 10, 13, 14, 15, 23
Elements	**permissions.checkSelfPermission(context, Camera):** A function inside of android phones that returns a permission's status.
	permissions.requestPermission(): A function inside of android phones that requests for permission to use certain services with a pop-up.
	1. toggleGPS: A method that allows the **User** to toggle the **App** permissions to retrieve **GPS Location**.
	2. toggleCamera: A method that allows the **User** to toggle the **App** permissions to access the **Camera**.
	3. updateUsername: A method that allows the **User** to update their username.
	4. updatePassword: A method that allows the **User** to update their password.
	logout: A method that allows the **User** to log out of the **App**.
Referenced By	**1.2.1.16.2 Fragment Inheritance Map**
Viewpoint	Pseudocode

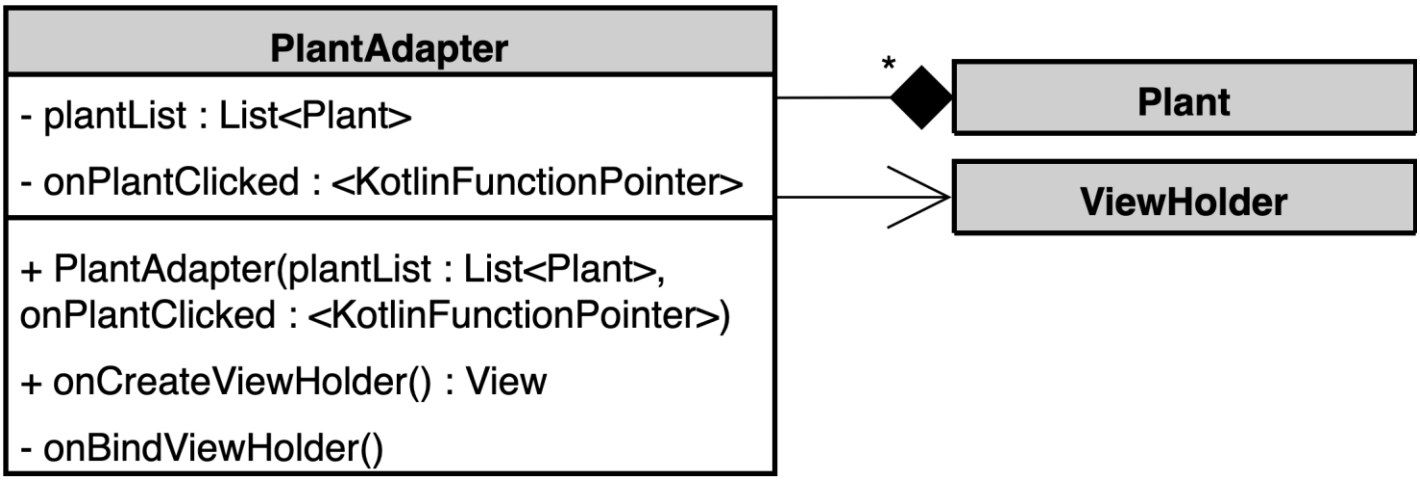
View 1.2.1.16.3: List Class Diagram View



Name	#1.2.1.16.3 List Class Diagram View
Purpose	Serves as an overview of the List class, which extends the Fragment base class.
Description	Class that represents the major list screens for the **App**.
Requirements	4, 6, 7, 10, 13, 14, 15, 23
Elements	**onCreateView:** A lifecycle method in each class that is called to inflate the fragment's layout and the UI components are set up. (https://developer.android.com/reference/android/app/Fragment#onCreateView(android.view.LayoutInflater,%20android.view.ViewGroup,%20android.os.Bundle))
	onViewCreated: A lifecycle method in each class that is called after onCreateView and performs the actual logic for the **User** interacting with the UI elements. (https://developer.android.com/reference/android/app/Fragment#onViewCreated(android.view.View,%20android.os.Bundle)
	requestor: A helper class that will handle communication with the **Server**.
	List::recyclerView: An object of the RecyclerView class that will hold and display the list. (https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView)
	1.2.1.16.3.1 List::plantAdapter: An object of the PlantAdapter class that will bind the **Plant** data to the RecyclerView.
	List::setUpList: A method that will set up the UI elements for the list.
	1.2.1.16.3.2 List::onPlantClicked: A method that will give the **User** the option to load a PlantInfo fragment when the **User** taps on a **Plant** on the list.
	Checklist::setUpList: A method that will set up the UI elements for a checklist.
	1.2.1.16.3.2 Checklist:onPlantClicked: A method that will give the **User** the option to view the **Plant's** info, remove the **Plant** from the checklist, or check off the **Plant**.
	Checklist::checkOffPlant: A method that will complete the checking off of a **Plant** by adding it to the **Collected Plants List**.
Referenced By	**1.2.1.16 Fragment Inheritance Map**

Viewpoint	Class Diagram

View 1.2.1.16.3.1: Plant Adapter Class Diagram View



Name	#1.2.1.16.3.1 Plant Adapter Class Diagram View
Purpose	Responsible for binding a list of **Plant** objects to a RecyclerView.
Description	The PlantAdapter class connects a list of **Plant** objects to a RecyclerView. It handles creating and binding views for each **Plant** and includes a way to define what happens when a **User** clicks on a **Plant**.
Requirements	4, 6, 10, 11, 14, 15
Elements	**plantList:** A list of **Plant** objects which will be ViewHolder objects managed by a RecyclerView.
	onPlantClicked: A lambda function that is designed to be passed in when the PlantAdapter is instantiated, defines what happens when a **Plant** is tapped on.
	onCreateViewHolder: This method is responsible for creating new ViewHolder objects, inflating the layout for each item in the list.
	onBindViewHolder: This method binds data from a specific **Plant** object to the ViewHolder, updating the UI for that item in the list.
Referenced By	**1.2.1.16.3 List Class Diagram View**
Viewpoint	Class Diagram

View 1.2.1.16.3.1.1: Plant Adapter Pseudocode View

```
1.
onBindViewHolder(holder, position)
     plant ← plantList[position]
     PUT plant.name
     PUT plant.type
     plantButton.onClick ← onPlantClicked(plant)
END
```

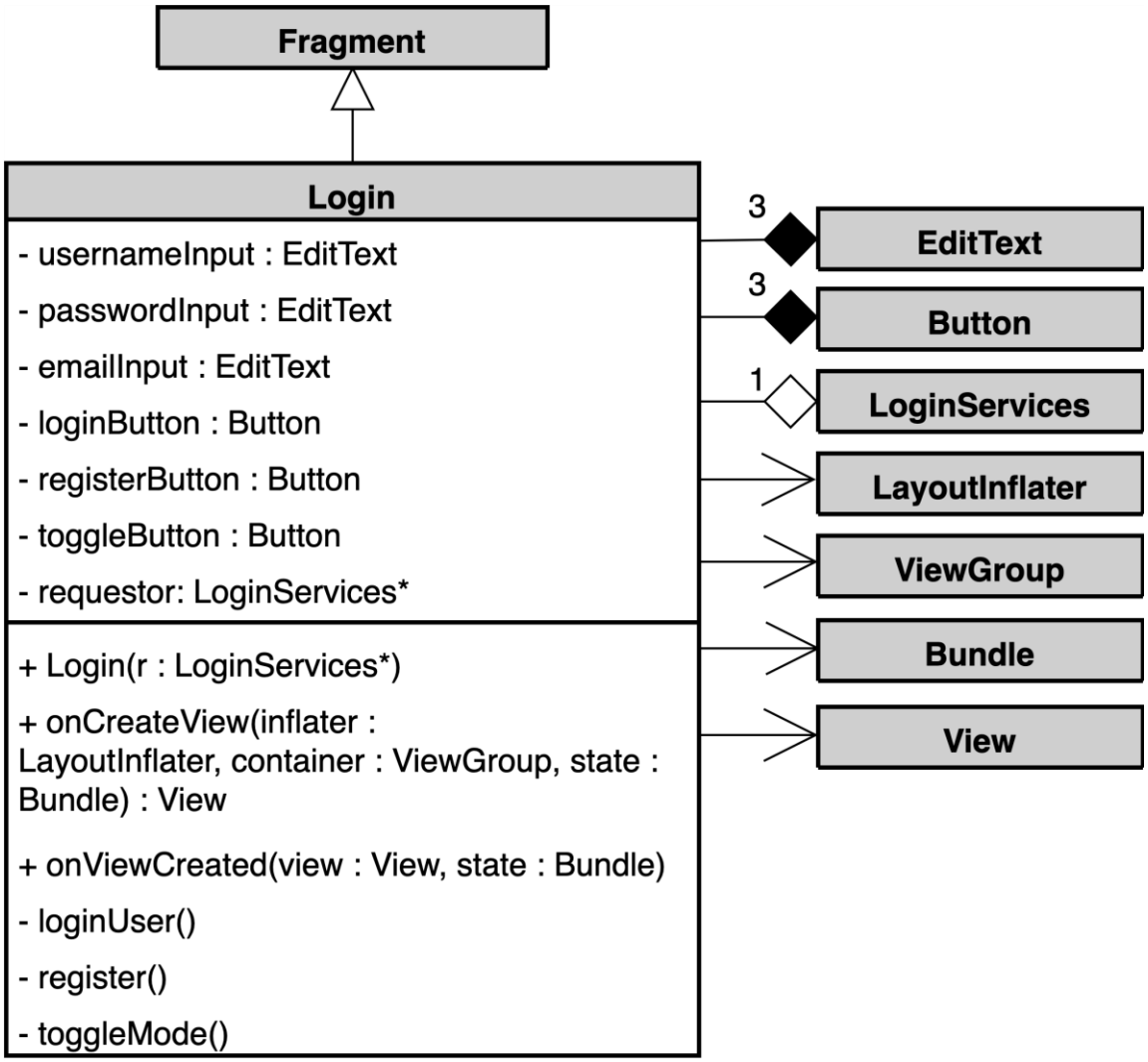
Name	#1.2.1.16.3.1.1 Plant Adapter Pseudocode view
Purpose	To outline the key methods that handle the **User** input while viewing the **All Plants List**, **Local Plants List, Collected Plant List**, or one of their **Gather Lists**.
Description	Provides pseudocode for interactions with individual **Plants** items displayed to the **User**.
Requirements	4, 6, 7, 10, 13, 14, 15, 23
Elements	**onBindViewHolder:** This method binds data from a specific **Plant** object to the ViewHolder, updating the UI for that item in the list.
Referenced By	**1.2.1.16.3.1 Plant Adapter Diagram**
Viewpoint	Pseudocode

View 1.2.1.16.3.2: List Pseudocode View

```
1.
List::onPlantClicked(plant)
     options ← [“View Plant Info”]
     alert ← new AlertDialog
     alert.setTitle(“Choose an action”)
     alert.setItems(options)
     IF user selects “View Plant Info”
          plantInfoFragment ← PlantInfo(requestor, plant)
          parentFragmentManager.beginTransaction()
          fragmentManager.replace(currentFragment, plantInfoFragment)
     alert.show()
END

2.
Checklist::onPlantClicked(plant)
     options ← [“View Plant Info”, “Check Off Plant”, “Remove plant”]
     alert ← new AlertDialog
     alert.setTitle(“Choose an action”)
     alert.setItems(options)
     IF user selects “View Plant Info”
          plantInfoFragment ← PlantInfo(requestor, plant)
          parentFragmentManager.beginTransaction()
          fragmentManager.replace(currentFragment, plantInfoFragment)
     IF user selects “Check off Plant”
          checkOffPlant(plant.getPlantId())
     IF user selects “Remove Plant”
          removePlantFromList(plant.getPlantId())
     alert.show()
END
```

Name	#1.2.1.16.3.2 List Pseudocode view
Purpose	To outline the key methods that handle the **User** input while viewing the **All Plants List**, **Local Plants List, Collected Plant List**, or one of their **Gather Lists**.
Description	Provides pseudocode for interactions with individual **Plants** items displayed to the **User**.
Requirements	4, 6, 7, 10, 13, 14, 15, 23
Elements	**List::onPlantClicked:** Loads a PlantInfo fragment when the **User** taps on a **Plant**.
	Checklist::onPlantClicked: Gives the **User** the option to view the **Plant's** info, remove the **Plant** from the checklist, or check off the **Plant.**
Referenced By	**1.2.1.16.3 List Class Diagram**
Viewpoint	Pseudocode

View 1.2.1.16.4: Login Class Diagram View



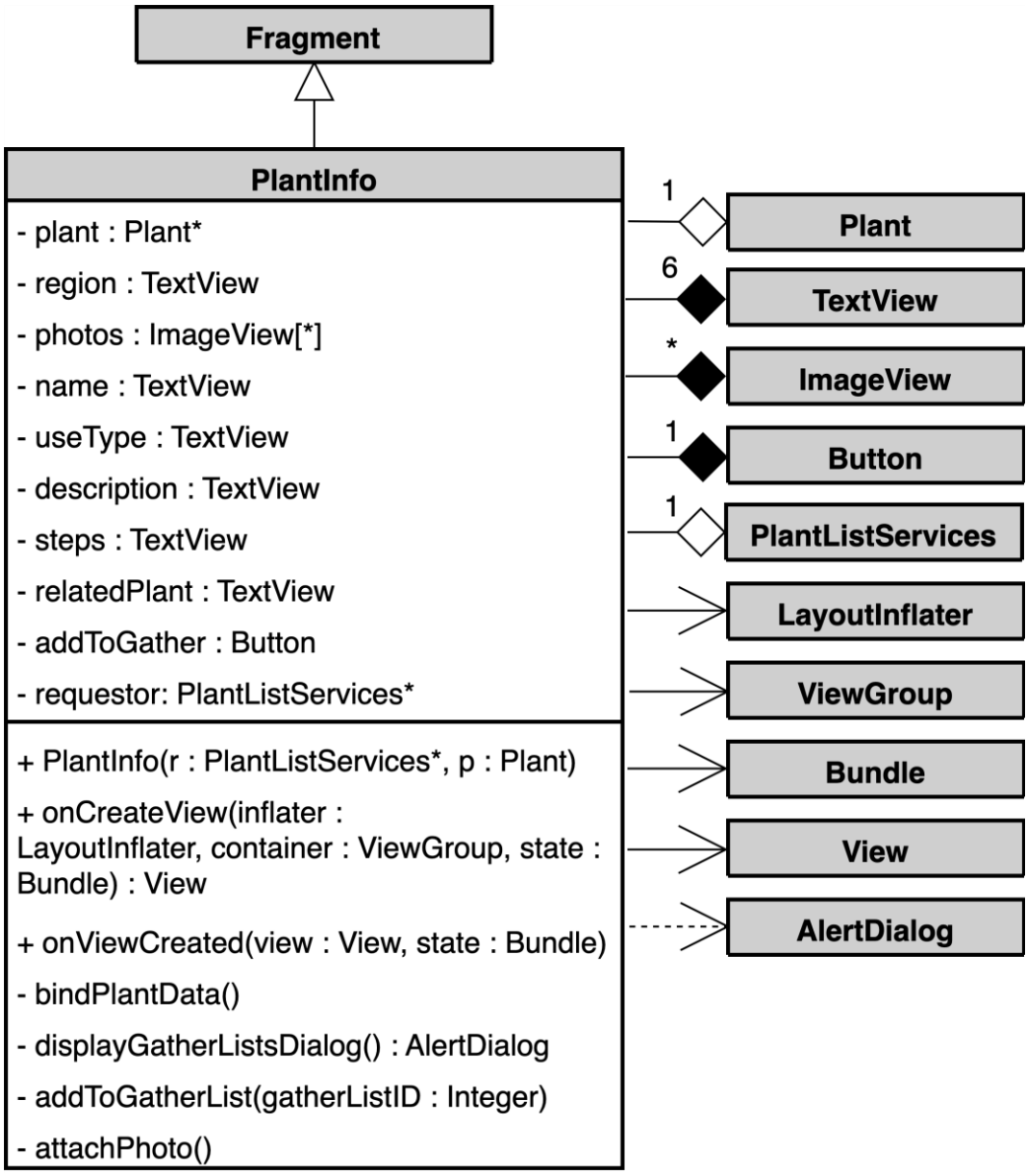
Name	1.2.1.16.4 Login Class Diagram View
Purpose	Serves as an overview of the Login class, which extends the Fragment base class.
Description	Class that represents the Login and Registration screens for the **App** as well as their methods and attributes.
Requirements	4, 6, 7, 10, 13, 14, 15, 23
Elements	**usernameInput:** An interface element for entering the username. (https://developer.android.com/reference/android/widget/EditText)
	passwordInput: An interface element for entering the password.
	emailInput: An interface element for entering the email.
	loginButton: An interface element the **User** can tap to initiate the login process. (https://developer.android.com/reference/android/widget/Button)
	registerButton: An interface element the **User** can tap to initiate the registration process.
	toggleButton: An interface element the **User** can tap to initiate the switch between login and registration views.
	requestor: A helper class that will handle communication with the **Server**.
	onCreateView: A lifecycle method in each class that is called to inflate the fragment's layout and the UI components are set up. https://developer.android.com/reference/android/app/Fragment#onCreateView(android.view.LayoutInflater,%20android.view.ViewGroup,%20android.os.Bundle)
	onViewCreated: A lifecycle method in each class that is called after onCreateView and performs the actual logic for the **User** interacting with the UI elements. https://developer.android.com/reference/android/app/Fragment#onViewCreated(android.view.View,%20android.os.Bundle)
	loginUser: A method that will carry out the login process.
	register: A method that will carry out the registration process.
	toggleMode: A method that will re-render the fragment to either display the necessary UI components for logging in or for registration.
Referenced By	**1.2.1.16 Overall Fragment Inheritance Map**
Viewpoint	Class Diagram

View 1.2.1.16.4.1: Login Class Pseudocode View

```
register()
     username ← get text from usernameInput
     password ← get text from passwordInput
     requestor.createAccount(username, password)
     loginUser()
```

Name	#1.2.1.16.4.1 Login Fragment Pseudocode View
Purpose	Shows the pseudocode of the register method.
Description	Shows the pseudocode for methods within the Login fragment. This fragment handles **User Authentication** and account creation.
Requirements	1, 4, 6, 7, 10, 13, 14, 15, 23
Elements	**loginUser:** A method that will carry out the login process.
	register: A method that will carry out the registration process.
	requestor.createAccount(username, password): sends a request to the server to create a new account so the user is not forgotten and required to create an account every time they want to use the app.
Referenced By	**1.2.1.16.4 Login Fragment Class Diagram**
Viewpoint	Pseudocode

View 1.2.1.16.5: PlantInfo Class Diagram View



Name	#1.2.1.16.5 PlantInfo Class Diagram View
Purpose	Serves as an overview of the PlantInfo class which extends the Fragment base class.
Description	Class that represents the screen for displaying an individual **Plant** on the **App** as well as its methods and attributes.
Requirements	4, 6, 7, 10, 13, 14, 15, 23
Elements	**1.2.1.8 plant:** A pointer to the current **Plant**, with its info being displayed as TextViews and ImageViews.
	addToGather: An interface element that will initiate the process of adding the **Plant** to a **Gather List**.
	requestor: A helper class that will handle communication with the **Server**.
	onCreateView: A lifecycle method in each class that is called to inflate the fragment's layout and the UI components are set up. (https://developer.android.com/reference/android/app/Fragment#onCreateView(android.view.LayoutInflater,%20android.view.ViewGroup,%20android.os.Bundle))
	onViewCreated: A lifecycle method in each class that is called after onCreateView and performs the actual logic for the **User** interacting with the UI elements. (https://developer.android.com/reference/android/app/Fragment#onViewCreated(android.view.View,%20android.os.Bundle)
	bindPlantData: A method that is responsible for updating the fragment's UI elements by binding the properties of a **Plant** object to the corresponding views in the layout.
	displayGatherListsDialog: A method that will display a button for each of the **User's Gather Lists.** (https://developer.android.com/reference/android/app/AlertDialog)
	addToGatherList: A method that will add the **Plant** to the requested **Gather List.**
	attachPhoto: A method that handles the **User** wanting to add a **Photo** to a **Plant** using the requestor.
Referenced By	**1.2.1.16 Overall Fragment Inheritance Map**
Viewpoint	Class Diagram

View 1.2.1.16.5.1: PlantInfo Pseudocode View

```
bindPlantData()
     pId ← plant.getPlantId()
     requestReturn ← requestor.getPlantInfo(pId)
     region ← requestReturn.region
     photos ← requestReturn.photos
     name ← requestReturn.name
     useType ← requestReturn.useType
     description ← requestReturn.description
     steps ← requestReturn.steps
     relatedPlant ← requestReturn.relatedPlant
```

Name	#1.2.1.16.5.1 PlantInfo Fragment Pseudocode View
Purpose	Shows the pseudocode of the PlantInfo fragment.
Description	Shows the pseudocode for methods within the **PlantInfo** fragment. This fragment displays the attributes of a **Plant**.
Requirements	6, 11
Elements	**getPlantInfo(plantId):** is a function that accesses the **Server** and returns all plant info related to a **Plant Id**.
	getPlantId(): is a function inside the plant class that returns its related Id for use.
Referenced By	**1.2.1.16.5 PlantInfo Class Diagram**
Viewpoint	Pseudocode

View 1.2.1.17: JSON Schema For Requests

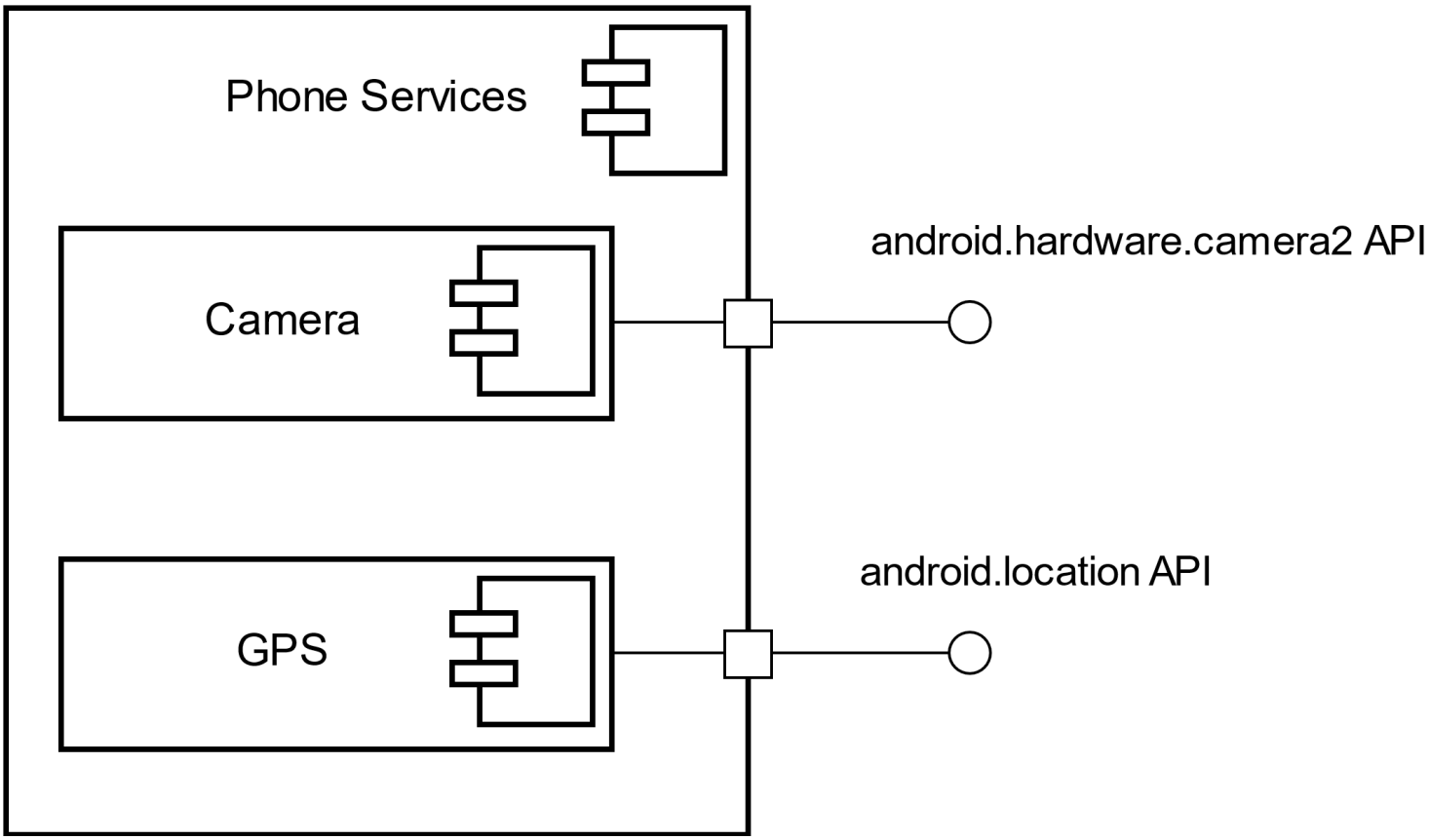
```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "auth": {
      "type": "object",
      "properties": {
        "jwt": {
          "type": "string",
          "description": "JSON Web Token for authentication"
        }
      },
      "required": ["jwt"]
    },
    "request": {
      "type": "object",
      "properties": {
        "operation": {
          "type": "string",
          "enum": ["CREATE", "UPDATE", "INSERT", "DELETE", "SELECT"],
          "description": "The type of operation being performed"
        },
        "resource": {
          "type": "string",
          "enum": [
            "Account",
            "Gather_List",
            "Collected_Plants_List",
            "Plant",
            "Plant_has_Region",
            "Gather_List_has_Plant",
            "User_has_Gather_List"
          ],
          "description": "The target resource for the operation"
        },
        "parameters": {
          "type": "object",
          "properties": {
            "userId": {
              "type": "integer",
              "description": "ID of the user"
            },
            "plantId": {
              "type": "integer",
              "description": "ID of the plant"
            },
            "regionId": {
              "type": "integer",
```

```
              "description": "ID of the region"
            },
            "gatherListId": {
              "type": "integer",
              "description": "ID of the gather list"
            }
          },
          "additionalProperties": false
        },
        "data": {
          "type": "object",
          "properties": {
            "username": {
              "type": "string",
              "description": "User's username"
            },
            "password": {
              "type": "string",
              "description": "User's password"
            },
            "email": {
              "type": "string",
              "format": "email",
              "description": "Recovery email for the user"
            },
            "userId": {
              "type": "integer",
              "description": "ID of the user for operations requiring explicit
user targeting"
            },
            "plantId": {
              "type": "integer",
              "description": "ID of the plant to be added/removed"
            }
          },
          "additionalProperties": false
        }
      },
      "required": ["operation", "resource"]
    }
  },
  "required": ["auth", "request"],
  "additionalProperties": false
}
```

Name	#1.2.1.17 Requests JSON Schema View
Purpose	To illustrate what the JSON will look like for requests being made to the **Server**.
Description	This JSON schema outlines how requests to the **Server** should be structured, including actions like creating or updating accounts and managing **Plant** lists. It ensures the necessary information is included for secure operations.
Requirements	3, 4, 7, 10, 11, 14-18, 21
Elements	**auth:** Contains authentication details required for the request to be valid.
	jwt: A string representing the JWT used for authentication.
	request: The main object that holds the details of the request, including operation and data.
	operation: Specifies the action to be performed.
	resource: Identifies the table being acted upon.
	parameters: Contains optional parameters necessary for certain requests.
	userId: An identifier for the **User** involved in the request.
	plantId: An identifier for a **Plant**, used in requests involving **Plant** data.
	regionId: An identifier for a region, used in requests involving **Plant** and location data.
	gatherListId: An identifier for a specific **Gather List**.
	data: Contains the actual data being sent in the request.
	username: The **User's** username, included in requests related to account creation or updates.
	password: The **User's** password, used in account-related requests to set or update passwords.
	email: An email address used for recovering the account, included in account creation or updates.
Referenced By	**1.1 Server View, 1.1.1.1 Update Gather List View, 1.1.1.2 Checking Off From Gather List View, 1.1.1.3 Get Gather/Collected List View, 1.1.2b Server Application View, 1.1.2.5 User Sign Up View, 1.2.1.1b User Login Process Data**

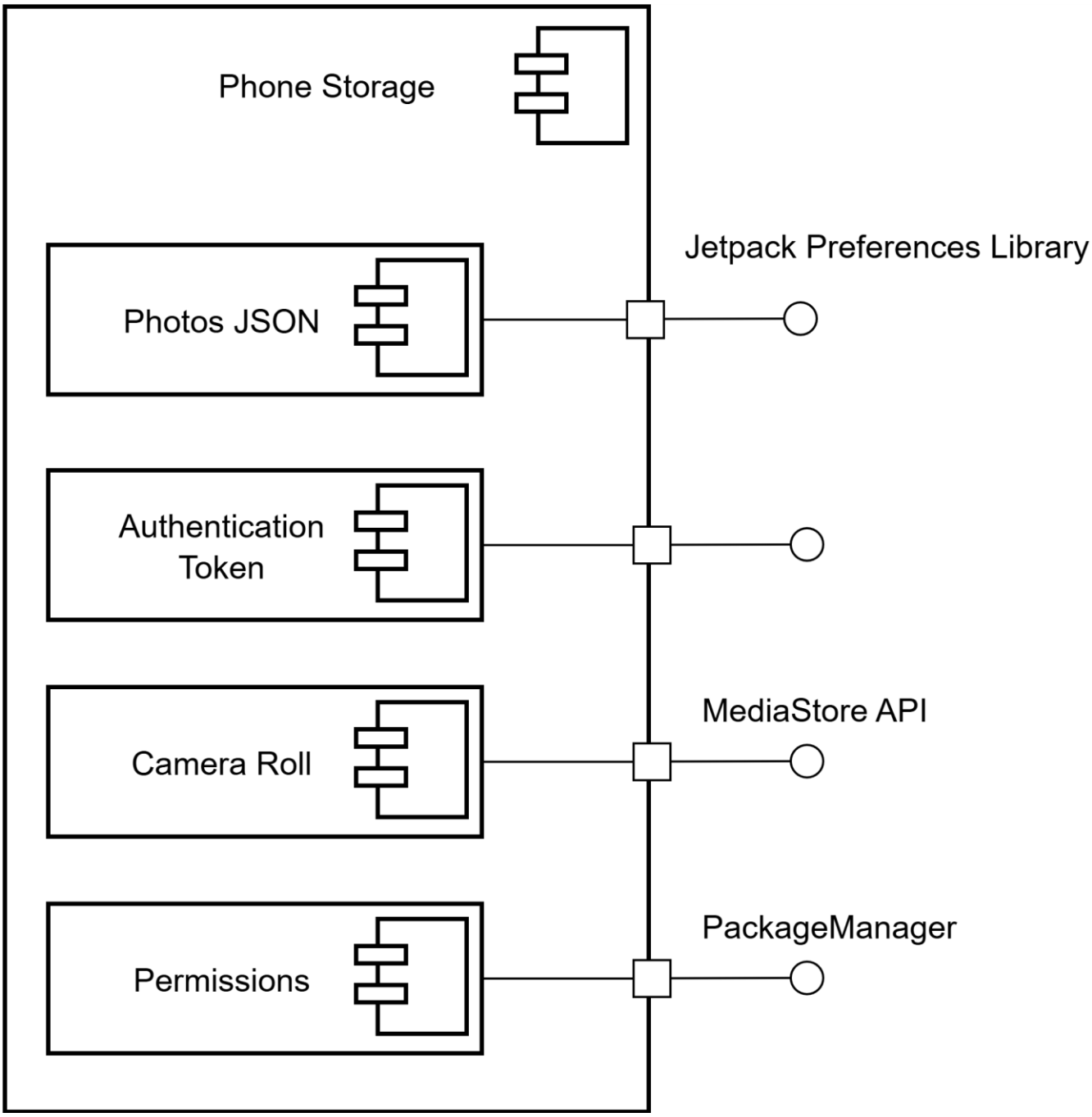
	Flow Diagram, 1.1.2.5 User Sign Up View, 1.2.1.7.2 Request Local Plants View, 1.2.1.7.3 Request All Plants View, 1.2.3.4a Send Authentication Token
Viewpoint	JSON Schema

View 1.2.2: Phone Services View



Name	#1.2.2 Phone Services View
Purpose	To manage the usage of device specific components. This component allows the client application direct access to **Phone** hardware.
Description	Device-specific components, which are the GPS and camera. The camera will be accessed through the Camera2 API under android.hardware.camera2. The GPS will be accessed through the android.location API.
Requirements	2, 12
Elements	**1.2.2 Phone Services:** Device-specific components, which include the GPS and camera.
	Camera: The **Photo** taking device.
	android.hardware.camera2 API: Android's built in API to handle camera services. https://developer.android.com/media/camera/camera2
	GPS: The Global Positioning System of the **Phone.**
	android.location API: Android's built in API to handle GPS services. https://developer.android.com/reference/android/location/package-summary
Referenced By	**1 Overall Application View, 1.2.1.7.2 Request Local Plants View**
Viewpoint	Component Diagram

View 1.2.3: Phone Storage View



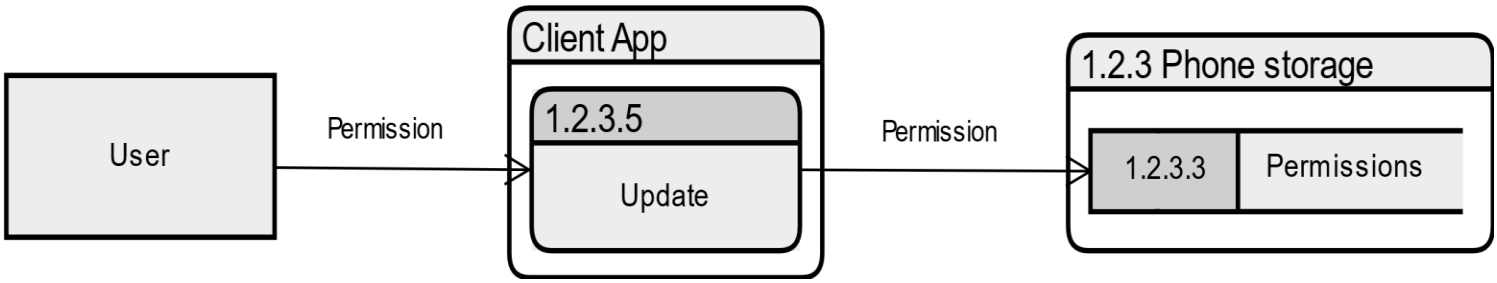
Name	# 1.2.3 Phone Storage View
Purpose	To manage which **Photos** on the **User's** device are attached to each **Plant.** This component will also store which **Permissions** the **User** has given and their **Session Token.**
Description	This view includes **User**-specific data storage elements, such as **Photos** JSON and media (Camera Roll). These are accessed locally and managed through the Jetpack Preferences Library for **Photos** JSON and MediaStore API for the camera roll. The **Phone** storage will also store a web token for the **User**. Permissions will be stored using Android's PackageManager.
Requirements	12, 13, 21, 23, 24
Elements	**1.2.3 Phone Storage:** Handles the overall file system on the **User's** device.
	1.2.3.1 Photos JSON: Keeps track of which **User Photos** are linked to each **Plant**.
	1.2.3.4 Authentication Token: Will store an **Encoded JWT.**
	1.2.3.2 Camera Roll: Manages **User** media files (images) through the MediaStore API.
	1.2.3.3 Permissions: Stores the location permissions in the **Phone's** operating system using the PackageManager.
	Jetpack Preferences Library: Android library used to edit and save **User Settings**. (https://developer.android.com/jetpack/androidx/releases/preference/)
	MediaStore API: Android's built-in API to handle images. (https://developer.android.com/reference/android/provider/MediaStore)
	Package Manager: Android API for retrieving information related to the application packages. **(**https://developer.android.com/reference/android/content/pm/PackageManager)
Referenced By	**1 Overall Application View, 1.2.1.7.2 Get Local Plants View**
Viewpoint	Component Diagram

View 1.2.3.1: Plants JSON Schema View

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "Plants": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "Id": {
            "type": "integer"
          },
          "Photos": {
            "type": "array",
            "items": {
              "type": "string"
            },
            "minItems": 0
          }
        },
        "required": ["Id", "Photos"],
        "additionalProperties": false
      }
    }
  },
  "required": ["Plants"],
  "additionalProperties": false
}
```

Name	#1.2.3.1 Plants JSON Schema View
Purpose	To illustrate what the JSON will look like that links **Plants** to **User Photos**.
Description	The **Photos** JSON is stored locally on the **Phone**. This JSON keeps track of which **Plants** go with each **Photo/s**. The properties of each **Plant** will be id and filenames of **Photos**.
Requirements	13
Elements	**JSON Object:** Represents **Plants** that have saved **Photo/s** referenced by an **Id**.
Referenced By	**1.2.1c User Photo Data Flow Diagram, 1.1.2.5 User Sign Up View, 1.2.3 Phone Storage View**
Viewpoint	JSON Schema

View 1.2.3.2: User Permissions Update Data Flow Diagram



Name	#1.2.3.2 User Permissions update Data flow diagram
Purpose	Illustrates the process and flow of data when a **User** updates their permissions for the **App.** Permissions are not required for the **App** to function but add more features.
Description	A **User** changes the permissions for the **App.** The permissions are then recorded and stored in their phone.
Requirements	2, 12, 13
Elements	**1.2.3.5 Update:** Takes permission states from the **User** and records them in the **Users** phone.
	1.2.3.3 Permissions: Device permissions for the **Camera**, **Camera Roll**, and **GPS.** Saving handled by the operating system.
	Permission: An integer representing the state of each requested permission (PERMISSION_GRANTED if you have the permission, or PERMISSION_DENIED if not) (https://developer.android.com/reference/androidx/core/content/ContextCompat#checkSelfPermission(android.content.Context,java.lang.String))
Referenced By	**1.2.3 Phone Storage View**
Viewpoint	Data Flow Diagram

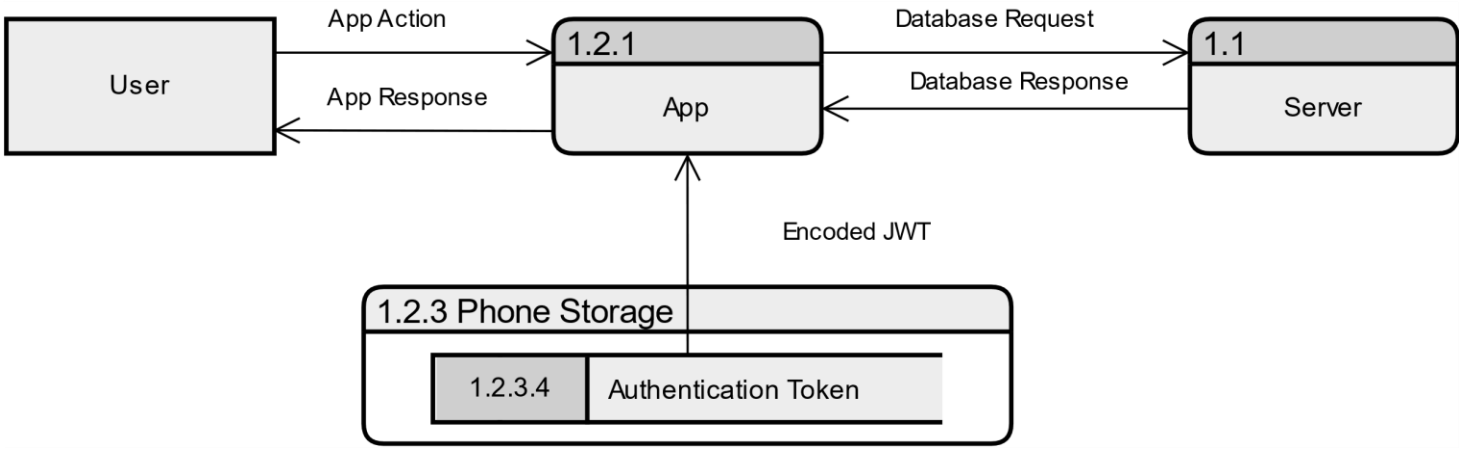
View 1.2.3.4: Authentication Token

Encoded JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

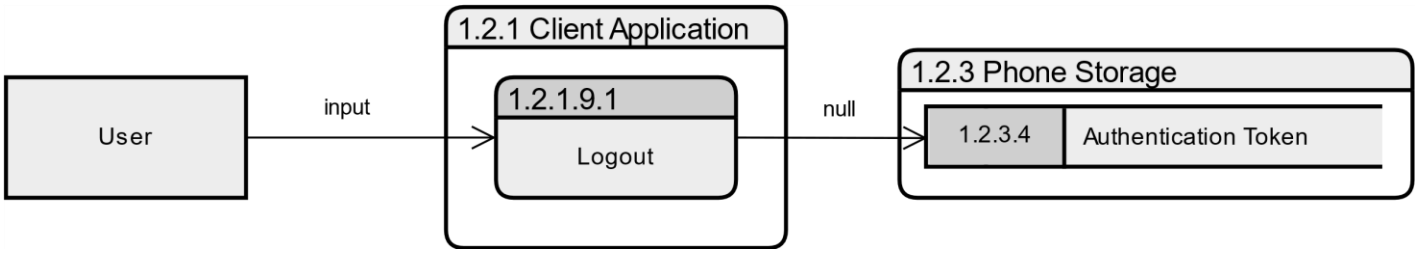
Name	# 1.2.3.4 Authentication Token
Purpose	Shows what the Authentication Token looks like.
Description	The Authentication Token is stored as an encoded Base64Url string.
Requirements	19
Elements	**Encoded JWT:** An encoded string used to authenticate the **User**. This is the form of the JWT in transit, and when sent to the **Client**.
Referenced By	**1.2.1.1c Authentication Token from Login, 1.1.2.5 User Sign Up View, 1.2.1b Logout Flowchart, 1.2.1.9 Request Services Class Diagram View**
Viewpoint	String

View 1.2.3.4a: Send Authentication Token



Name	#1.2.3.4a Send Authentication Token
Purpose	This shows the process of sending the JSON Web Token (JWT) to the **App** and to the **Server** whenever the **Database** needs to be accessed.
Description	The **User** will perform an action within the **App** that requires the **User** to be logged in. The **App** will retrieve the JWT from the **Phone's** storage system. The **App** will then send the JWT as part of the request to the **Server**. The JWT was created and stored during the login/signup process.
Requirements	21
Elements	**User**: A **Person** using the **App** who has been **Authenticated.**
	App: A **Mobile Application** serving as a **Client** for Forage. Retrieves the Authentication JSON from storage and sends it to the **Server**.
	1.1.2.2 Encoded JWT: An encoded string used to authenticate the **User**. This is the form of the JWT in transit, and when sent to the **Client**.
	1.2.3.4 Authentication Token: Where the **Encoded JWT** is stored within the device's storage system.
	App Action: Any **User** input that requires a response from the server.
	App Response: New information that is displayed to the **User** after receiving the response from the server.
	1.2.1.17 Database Request: The **User's** request as well as the Encoded JWT from storage, encrypted and then packaged and sent as an http request.
	1.1.1.6b Database Response: Http response to the **User's** request.
	Server: The back-end part of the **Client**-**Server** pair.
Referenced By	**1.2.1.1c Authentication Token from Login, 1.1.2.5 User Sign Up View, 1.2.1b Logout Flowchart, 1.2.1.9 Request Services Class Diagram View, 1.2.3.4 Authentication Token, 1.2.1.7.2 Request Local Plants View, 1.2.1.7.3 Request All Plants View, 1.1.1.6b Database Data JSON View**
Viewpoint	Data Flow Diagram

View 1.2.3.4b: Removal of the Authentication Token



Name	#1.2.3.4b Removal of the Authentication Token
Purpose	This diagram shows the flow of data as the JSON Web Token (JWT) is removed from the **Phone** storage upon the **User** logging out.
Description	The **User** will request to logout in the **App**. The **App** will process that request and send a null value to **Phone** storage to delete the stored JWT.
Requirements	21
Elements	**User**: A **Person** using the **App** who has been **Authenticated** and initiates the request to log out.
	1.2.1.9.1 Logout: The **App** will then send a null value to **Phone** storage to delete the stored Authentication JSON.
	1.2.3.4 Authentication Token: The **Encoded JWT** that is stored in the phone's storage system.
Referenced By	**1.2.1.1c Authentication Token from Login, 1.1.2.5 User Sign Up View, 1.2.1b Logout Flowchart, 1.2.1.9 Request Services Class Diagram View**
Viewpoint	Data Flow Diagram

View 1.2.3.5: Retrieve Photo Pseudocode View

```
retrievePhotos(plantId)
     READ photoJson
     FOR photoPath in photoJSON[plantId]
          plantPhotos.append(photoPath)
     RETURN plantPhotos
```

Name	#1.2.3.5 Retrieve Photo Pseudocode view
Purpose	This view is the pseudocode used to retrieve the locally stored **Photos** for the **App's** use.
Description	This function will be given the id of a **Plant**. It will then read the stored JSON and retrieve the file paths for all **Photos** tied to the given id. The file paths will be placed in a list and returned to the caller.
Requirements	7
Elements	**photoJSON:** A JSON that keeps track of which **User Photos** are linked to each **Plant**.
	photoPath: The file path for the related **Photo** in the **Phone's Camera Roll.**
	PlantId: The unique identifier of the **Plant** that we are retrieving **Photos** for.
Referenced By	**1.2.1c User Photo Data Flow Diagram, 1.2.1d User Photo Data Flow Diagram**
Viewpoint	Pseudocode

Additional Information

Requirement Traceability Matrix

Requirements Traceability Matrix

Requirement #:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
View 1: Overall Application	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
View 1.1: Server								■	■															
View 1.1.1: Database							■	■	■	■	■	■												
View 1.1.1.1 Update Gather List																	■	■						
View 1.1.1.2 Checking Off From Gather List															■				■					
View 1.1.1.3 Get Gather/Collected List														■		■								
View 1.1.1.5 Hashed Password																							■	
View 1.1.1.6a DatabaseDatabse Queries			■			■	■			■	■		■	■	■	■	■	■	■					
View 1.1.1.6b Database Data JSON			■			■	■			■	■			■	■	■	■	■	■	■		■	■	
View 1.1.2a: Server Application Component	■			■	■	■	■			■				■	■	■	■	■	■	■	■		■	
View 1.1.2b: Server Application								■	■											■	■	■	■	
View 1.1.2.0a: Router and Controller	■						■			■				■	■	■	■	■	■	■	■	■	■	
View 1.12.0b: Router Pseudocode			■			■	■			■	■		■	■	■	■	■	■	■					
View 1.1.2.1a: Token Authentication Class																				■		■		
View 1.1.2.1b: Token Authentication PseudocodePsuedocode																				■		■		
View 1.1.2.2 Controller Pseudocode			■	■																				
View 1.1.2.3 JSON Web Token																			■	■	■			
View 1.1.2.4a: Login AuthenticationAuthe Flowchart View																				■			■	
View 1.1.2.4b: Authentication View																					■	■		
View 1.1.2.5 User Sign Up																			■		■	■		
View 1.2.1a: Client Application Site Map			■	■	■	■	■			■	■	■	■	■	■	■	■	■	■					
View 1.2.1b: Overall User Flow Diagram										■									■	■				
View 1.2.1c: User Photo Data Flow Diagram							■						■											

View 1.2.1.16.5.1: Plant Info Pseudocode						■					■													
View 1.2.1.17: JSON Schema For Requests			■	■			■			■	■			■	■	■	■	■			■			
View 1.2.2: Phone Services View		■										■												
View 1.2.3: Phone Storage View												■	■								■		■	■
View 1.2.3.1: Plants JSON Schema													■											
View 1.2.3.2: User Permissions Update Data Flow Diagram		■										■	■											
View 1.2.3.4: Authentication Token																			■		■			
View 1.2.3.4a: Send Authentication Token																					■			
View 1.2.3.4b: Removal of the Authentication Token																					■			
View 1.2.3.5: Retrieve Photo Pseudocode							■																	
Requirement #:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24